

Marco Wiering and Martijn van Otterlo (Eds.)

Reinforcement Learning

Adaptation, Learning, and Optimization, Volume 12

Series Editor-in-Chief

Meng-Hiot Lim
Nanyang Technological University, Singapore
E-mail: emhlim@ntu.edu.sg

Yew-Soon Ong
Nanyang Technological University, Singapore
E-mail: asysong@ntu.edu.sg

Further volumes of this series can be found on our homepage: springer.com

Vol. 1. Jingqiao Zhang and Arthur C. Sanderson
Adaptive Differential Evolution, 2009
ISBN 978-3-642-01526-7

Vol. 2. Yoel Tenne and Chi-Keong Goh (Eds.)
Computational Intelligence in
Expensive Optimization Problems, 2010
ISBN 978-3-642-10700-9

Vol. 3. Ying-ping Chen (Ed.)
Exploitation of Linkage Learning in Evolutionary Algorithms, 2010
ISBN 978-3-642-12833-2

Vol. 4. Anyong Qing and Ching Kwang Lee
Differential Evolution in Electromagnetics, 2010
ISBN 978-3-642-12868-4

Vol. 5. Ruhul A. Sarker and Tapabrata Ray (Eds.)
Agent-Based Evolutionary Search, 2010
ISBN 978-3-642-13424-1

Vol. 6. John Seiffertt and Donald C. Wunsch
Unified Computational Intelligence for Complex Systems, 2010
ISBN 978-3-642-03179-3

Vol. 7. Yoel Tenne and Chi-Keong Goh (Eds.)
Computational Intelligence in Optimization, 2010
ISBN 978-3-642-12774-8

Vol. 8. Bijaya Ketan Panigrahi, Yuhui Shi, and Meng-Hiot Lim (Eds.)
Handbook of Swarm Intelligence, 2011
ISBN 978-3-642-17389-9

Vol. 9. Lijuan Li and Feng Liu
Group Search Optimization for Applications in Structural Design, 2011
ISBN 978-3-642-20535-4

Vol. 10. Jeffrey W. Tweedale and Lakhmi C. Jain
Embedded Automation in Human-Agent Environment, 2011
ISBN 978-3-642-22675-5

Vol. 11. Hitoshi Iba and Claus C. Aranha
Practical Applications of Evolutionary Computation to Financial Engineering, 2012
ISBN 978-3-642-27647-7

Vol. 12. Marco Wiering and Martijn van Otterlo (Eds.)
Reinforcement Learning, 2012
ISBN 978-3-642-27644-6

Marco Wiering and Martijn van Otterlo (Eds.)

Reinforcement Learning

State-of-the-Art



Editors

Dr. Marco Wiering
University of Groningen
The Netherlands

Dr. Ir. Martijn van Otterlo
Katholieke Universiteit Leuven
Heverlee
Belgium

ISSN 1867-4534
ISBN 978-3-642-27644-6
DOI 10.1007/978-3-642-27645-3
Springer Heidelberg New York Dordrecht London

e-ISSN 1867-4542
e-ISBN 978-3-642-27645-3

Library of Congress Control Number: 2011945323

© Springer-Verlag Berlin Heidelberg 2012

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

'Good and evil, reward and punishment, are the only motives to a rational creature: these are the spur and reins whereby all mankind are set on work, and guided.' (Locke)

Foreword

Reinforcement learning has been a subject of study for over fifty years, but its modern form—highly influenced by the theory of Markov decision processes—emerged in the 1980s and became fully established in textbook treatments in the latter half of the 1990s. In *Reinforcement Learning: State-of-the-Art*, Martijn van Otterlo and Marco Wiering, two respected and active researchers in the field, have commissioned and collected a series of eighteen articles describing almost all the major developments in reinforcement learning research since the start of the new millennium. The articles are surveys rather than novel contributions. Each authoritatively treats an important area of Reinforcement Learning, broadly conceived as including its neural and behavioral aspects as well as the computational considerations that have been the main focus. This book is a valuable resource for students wanting to go beyond the older textbooks and for researchers wanting to easily catch up with recent developments.

As someone who has worked in the field for a long time, two things stand out for me regarding the authors of the articles. The first is their youth. Of the eighteen articles, sixteen have as their first author someone who received their PhD within the last seven years (or who is still a student). This is surely an excellent sign for the vitality and renewal of the field. The second is that two-thirds of the authors hail from Europe. This is only partly due to the editors being from there; it seems to reflect a real shift eastward in the center of mass of reinforcement learning research, from North America toward Europe. Vive le temps et les différences!

October 2011

Richard S. Sutton

Preface

A question that pops up quite often among reinforcement learning researchers is on what one should recommend if a student or a colleague asks for

"some good and recent book that can introduce me to reinforcement learning".

The most important goal in creating this book was to provide at least a good answer to that question.

A Book about Reinforcement Learning

A decade ago the answer to our leading question would be quite easy to give; around that time two dominant books existed that were fully up-to-date. One is the excellent introduction¹ to reinforcement learning by Rich Sutton and Andy Barto from 1998. This book is written from an *artificial intelligence* perspective, has a great educational writing style and is widely used (around ten thousand citations at the time of writing). The other book was written by Dimitri Bertsekas and John Tsitsiklis in 1996 and was titled *neuro-dynamic programming*². Written from the standpoint of *operations research*, the book rigorously and in a mathematically precise way describes dynamic programming and reinforcement learning with a particular emphasis on approximation architectures. Whereas Sutton and Barto always maximize rewards, talk about *value functions*, *rewards* and are biased to the $\{V, Q, S, A, T, R\}$ part of the alphabet augmented with π , Bertsekas and Tsitsiklis talk about *cost-to-go-functions*, always minimize costs, and settle on the $\{J, G, I, U\}$ part of the alphabet augmented with the greek symbol μ . Despite these superficial (notation) differences, the distinct writing styles and backgrounds, and probably also the audience for which these books were written, both tried to give a thorough introduction

¹ Sutton and Barto, (1998) Reinforcement Learning: An Introduction, MIT Press.

² Bertsekas and Tsitsiklis (1996) *Neuro-Dynamic Programming*, Athena Scientific.

to this exciting new research field and succeeded in doing that. At that time, the big merge of insights in both operations research and artificial intelligence approaches to behavior optimization was still ongoing and many fruitful cross-fertilization happened. Powerful ideas and algorithms such as *Q*-learning and *TD*-learning had been introduced quite recently and so many things were still unknown.

For example, questions about *convergence* of combinations of algorithms and function approximators arose. Many theoretical and experimental questions about convergence of algorithms, numbers of required samples for guaranteed performance, and applicability of reinforcement learning techniques in larger intelligent architectures were largely unanswered. In fact, many new issues came up and introduced an ever increasing pile of research questions waiting to be answered by bright, young PhD students. And even though both Sutton & Barto and Bertsekas & Tsitsiklis were excellent at introducing the field and eloquently describing the underlying methodologies and issues of it, at some point the field grew so large that new texts were required to capture all the latest developments. Hence this book, as an attempt to fill the gap.

This book is the first book about reinforcement learning featuring only state-of-the-art surveys on the main subareas. However, we can mention several other interesting books that introduce or describe various reinforcement learning topics too. These include a collection³ edited by Leslie Kaelbling in 1996 and a new edition of the famous Markov decision process handbook⁴ by Puterman. Several other books^{5,6} deal with the related notion of *approximate dynamic programming*. Recently additional books have appeared on Markov decision processes⁷, reinforcement learning⁸, function approximation⁹ and relational knowledge representation for reinforcement learning¹⁰. These books just represent a sample of a larger number of books relevant for those interested in reinforcement learning of course.

³ L.P. Kaelbling (ed.) (1996) Recent Advances in Reinforcement Learning, Springer.

⁴ M.L. Puterman (1994, 2005) Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley.

⁵ J. Si, A.G. Barto, W.B. Powell and D. Wunsch (eds.) (2004) Handbook of Learning and Approximate Dynamic Programming, IEEE Press.

⁶ W.B. Powell (2011) Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition, Wiley.

⁷ O. Sigaud and O. Buffet (eds.) (2010) Markov Decision Processes in Artificial Intelligence, Wiley-ISTE.

⁸ C. Szepesvari (2010) Algorithms for Reinforcement Learning, Morgan-Claypool.

⁹ L. Busoniu, R. Babuska, B. De Schutter and D. Ernst (2010) Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press.

¹⁰ M. van Otterlo (2009) The Logic Of Adaptive Behavior, IOS Press.

Reinforcement Learning: A Field Becoming Mature

In the past one and a half decade, the field of reinforcement learning has grown tremendously. New insights from this recent period – having much to deal with richer, and firmer, theory, increased applicability, scaling up, and connections to (probabilistic) artificial intelligence, brain theory and general adaptive systems – are not reflected in any recent book. Richard Sutton, one of the founders of modern reinforcement learning described¹¹ in 1999 three distinct areas in the development of reinforcement learning; *past, present and future*.

The RL *past* encompasses the period until approximately 1985 in which the idea of *trial-and-error* learning was developed. This period emphasized the use of an active, exploring agent and developed the key insight of using a scalar reward signal to specify the *goal* of the agent, termed the *reward hypothesis*. The methods usually only learned policies and were generally incapable of dealing effectively with delayed rewards.

The RL *present* was the period in which *value functions* were formalized. Value functions are at the heart of reinforcement learning and virtually all methods focus on approximations of value functions in order to compute (optimal) policies. The *value function hypothesis* says that approximation of value functions is the dominant purpose of intelligence.

At this moment, we are well underway in the reinforcement learning *future*. Sutton made predictions about the direction of this period and wrote "*Just as reinforcement learning present took a step away from the ultimate goal of reward to focus on value functions, so reinforcement learning future may take a further step away to focus on the structures that enable value function estimation [...] In psychology, the idea of a developing mind actively creating its representations of the world is called **constructivism**. My prediction is that for the next tens of years reinforcement learning will be focused on constructivism.*" Indeed, as we can see in this book, many new developments in the field have to do with new structures that enable value function approximation. In addition, many developments are about *properties, capabilities and guarantees* about convergence and performance of these new structures. Bayesian frameworks, efficient linear approximations, relational knowledge representation and decompositions of hierarchical and multi-agent nature all constitute new structures employed in the reinforcement learning methodology nowadays.

Reinforcement learning is currently an established field usually situated in machine learning. However, given its focus on behavior learning, it has many connections to other fields such as psychology, operations research, mathematical optimization and beyond. Within artificial intelligence, there are large overlaps with probabilistic and decision-theoretic planning as it shares many goals with the planning community (e.g. the international conference on automated planning systems, ICAPS). In very recent editions of the international planning competition (IPC), methods originating from the reinforcement learning literature have entered the

¹¹ R.S. Sutton (1999) Reinforcement Learning: Past, Present, Future – SEAL'98.

competitions and did very well, in both probabilistic planning problems, and a recent "learning for planning" track.

Reinforcement learning research is published virtually everywhere in the broad field of artificial intelligence, simply because it is both a general methodology for behavior optimization as well as a set of computational tools to do so. All major artificial intelligence journals feature articles on reinforcement learning nowadays, and have been doing so for a long time. Application domains range from robotics and computer games to network routing and natural language dialogue systems and reinforcement learning papers appear at fora dealing with these topics. A large portion of papers appears every year (or two years) at the established top conferences in artificial intelligence such as IJCAI, ECAI and AAAI, and many also at top conferences with a particular focus on statistical machine learning such as UAI, ICML, ECML and NIPS. In addition, conferences on artificial life (Alife), adaptive behavior (SAB), robotics (ICRA, IROS, RSS) and neural networks and evolutionary computation (e.g. IJCNN and ICANN) feature much reinforcement learning work. Last but not least, in the past decade many specialized reinforcement learning workshops and tutorials have appeared at all the major artificial intelligence conferences.

But even though the field has much to offer to many other fields, and reinforcement learning papers appear everywhere, the current status of the field renders it natural to introduce fora with a specific focus on reinforcement learning methods. The *European workshop on reinforcement learning* (EWRL) has gradually become one such forum, growing every two years considerably and most recently held in Nancy (2008) and co-located with ECML (2011). Furthermore, the *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning* (ADPRL) has become yet another meeting point for researchers to present and discuss their latest research findings. Together EWRL and ADPRL show that the field has progressed a lot and requires its own community and events.

Concerning practical aspects of reinforcement learning, and more importantly, concerning benchmarking, evaluation and comparisons, much has happened. In addition to the planning competitions (e.g. such as the IPC), several editions of the reinforcement learning competitions¹² have been held with great success. Contestants competed in several classic domains (such as pole balancing) but also new and exciting domains such as the computer games Tetris and Super Mario. Competitions can promote code sharing and reuse, establish benchmarks for the field and be used to evaluate and compare methods on challenging domains. Another initiative for promoting more code and solution reuse is the RL-Glue framework¹³, which provides an abstract reinforcement learning framework that can be used to share methods and domains among researchers. RL-Glue can connect to most common programming languages and thus provides a system- and language-independent software framework for experimentation. The competitions and RL-Glue help to further mature the field of reinforcement learning, and enable better scientific methods to test, compare and reuse reinforcement learning methods.

¹² <http://www.rl-competition.org/>

¹³ glue.rl-community.org/

Goal of the Book and Intended Audience

As said before, we have tried to let this book be an answer to the question "*what book would you recommend to learn about current reinforcement learning?*". Every person who could pose this question is contained in the potential audience for this book. This includes PhD and master students, researchers in reinforcement learning itself, and researchers in any other field who want to know about reinforcement learning. Having a book with 17 surveys on the major areas in current reinforcement learning provides an excellent starting point for researchers to continue expanding the field, applying reinforcement learning to new problems and to incorporate principled behavior learning techniques in their own intelligent systems and robots.

When we started the book project, we first created a long list of possible topics and grouped them, which resulted in a list of almost twenty large subfields of reinforcement learning in which many new results were published over the last decade. These include established subfields such as *evolutionary reinforcement learning*, but also newer topics such as *relational knowledge representation approaches* and *Bayesian frameworks* for learning and planning. *Hierarchical approaches*, about which a chapter is contained in this book, form the first subfield that basically emerged¹⁴ right after the appearance of two mentioned books, and for that reason, were not discussed at that time.

Our philosophy when coming up with this book was to let the pool of authors reflect the youth and the active nature of the field. To that end, we selected and invited mainly young researchers in the start of their careers. Many of them finished their PhD studies in recent years, and that ensured that they were active and expert in their own sub-field of reinforcement learning, full of ideas and enthusiastic about that sub-field. Moreover, it gave them an excellent opportunity to promote that sub-field within the larger research area. In addition, we also invited several more experienced researchers who are recognized for their advances in several subfields of reinforcement learning. This all led to a good mix between different views on the subject matter. The initial chapter submissions were of very high quality, as we had hoped for. To complete the whole quality assurance procedure, we – the editors – together with a group of leading experts as reviewers, provided at least three reviews for each chapter. The results were that chapters were improved even further and that the resulting book contains a huge number of references to work in each of the subfields.

The resulting book contains 19 chapters, of which one contains introductory material on reinforcement learning, dynamic programming, Markov decision processes and foundational algorithms such as *Q*-learning and value iteration. The last chapter reflects on the material in the book, discusses things that were left out, and points out directions for further research. In addition, this chapter contains personal reflections and predictions about the field. The 17 chapters that form the core of the book are each self-contained introductions and overviews of subfields of reinforcement

¹⁴ That is not to say that there were no hierarchical approaches, but the large portion of current hierarchical techniques appeared after the mid-nineties.

learning. In the next section we will give an overview of the structure of the book and its chapters. In total, the book features 30 authors, from many different institutes and different countries.

The Structure of This Book

The book consists of 18 surveys of sub-fields of reinforcement learning which are grouped together in four main categories which we will describe briefly in the following. The first chapter, **Reinforcement Learning and Markov Decision Processes** by *Martijn van Otterlo and Marco Wiering*, contains introductory material on basic concepts and algorithms. It discusses Markov decision processes and model-based and model-free algorithms for solving them. The goal of this chapter is to provide a quick overview of what constitute the main components of any reinforcement learning method, and it provides the necessary background for all other chapters. All surveys were written assuming this background was provided beforehand. The last chapter of the book, **Conclusions, Future Directions and Outlook** by *Marco Wiering and Martijn van Otterlo*, reflects on the material in the chapters and lists topics that were not discussed and directions for further research. In addition, it contains a list of personal reflections and predictions on the field, in the form of short statements written by several authors of chapters in the book. The main part of the book contains four groups of chapters and we will briefly introduce them individually in the following.

EFFICIENT SOLUTION FRAMEWORKS

The first part of the book contains several chapters on modern solution frameworks used in contemporary reinforcement learning. Most of these techniques can be understood in the light of the framework defined in the introduction chapter, yet these new methods emphasize more sophisticated use of samples, models of the world, and much more.

The first chapter in this part, **Batch Reinforcement Learning** by *Sascha Lange, Thomas Gabel, and Martin Riedmiller* surveys techniques for *batch learning* in the context of value function approximation. Such methods can make use of highly optimized regression techniques to learn robust and accurate value functions from huge amounts of data. The second chapter, **Least-Squares Methods for Policy Iteration** by *Lucian Buşoniu, Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos, Robert Babuška, and Bart De Schutter* surveys a recent trend in reinforcement learning on robust linear approximation techniques for policy learning. These techniques come with a solid set of mathematical techniques with which one can establish guarantees about learning speed, approximation accuracy and bounds. The third chapter, **Learning and Using Models** by *Todd Hester and Peter Stone* describes many ways in which models of the world can be learned and how they can speed up reinforcement learning. Learned models can be used for more efficient value updates, for planning, and for more effective exploration. World models

represent general knowledge about the world and are, because of that, good candidates to be transferred to other, related tasks. More about the transfer of knowledge in reinforcement learning is surveyed in the chapter **Transfer in Reinforcement Learning: a Framework and a Survey** by Alessandro Lazaric. When confronted with several related tasks, various things can, once learned, be reused in a subsequent task. For example, policies can be reused, but depending on whether the state and/or action spaces of the two related tasks differ, other methods need to be applied. The chapter not only surveys existing approaches, but also tries to put them in a more general framework. The remaining chapter in this part, **Sample Complexity Bounds of Exploration** by Lihong Li surveys techniques and results concerning the sample complexity of reinforcement learning. For all algorithms it is important to know how many samples (examples of interactions with the world) are needed to guarantee a minimal performance on a task. In the past decade many new results have appeared that study this vital aspect in a rigorous and mathematical way and this chapter provides an overview of them.

CONSTRUCTIVE-REPRESENTATIONAL DIRECTIONS

This part of the book contains several chapters in which either *representations* are central, or their construction and use. As mentioned before, a major aspect of constructive techniques are the structures that enable value function approximation (or policies for that matter). Several major new developments in reinforcement learning are about finding new representational frameworks to learn behaviors in challenging new settings.

In the chapter **Reinforcement Learning in Continuous State and Action Spaces** by Hado van Hasselt many techniques are described for problem representations that contain continuous variables. This has been a major component in reinforcement learning for a long time, for example through the use of neural function approximators. However, several new developments in the field have tried to either more rigorously capture the properties of algorithms dealing with continuous states and actions or have applied such techniques in novel domains. Of particular interest are new techniques for dealing with continuous actions, since this effectively renders the amount of applicable actions infinite and requires sophisticated techniques for computing optimal policies. The second chapter, **Solving Relational and First-Order Logical Markov Decision Processes: A Survey** by Martijn van Otterlo describes a new representational direction in reinforcement learning which started around a decade ago. It covers all representations strictly more powerful than propositional (or; attribute-value) representations of states and actions. These include modelings as found in logic programming and first-order logic. Such representations can represent the world in terms of objects and relations and open up possibilities for reinforcement learning in a much broader set of domains than before. These enable many new ways of generalization over value functions, policies and world models and require methods from logical machine learning and knowledge representation to do so. The next chapter, **Hierarchical Approaches** by Bernhard Hengst too surveys a representational direction, although here representation refers to the structural decomposition of a *task*, and with that implicitly of the underlying Markov decision

processes. Many of the hierarchical approaches appeared at the end of the nineties, and since then a large number of techniques has been introduced. These include new decompositions of tasks, value functions and policies, and many techniques for automatically learning task decompositions from interaction with the world. The final chapter in this part, **Evolutionary Computation for Reinforcement Learning** by *Shimon Whiteson* surveys evolutionary search for good policy structures (and value functions). Evolution has always been a good alternative for iterative, incremental reinforcement learning approaches and both can be used to optimize complex behaviors. Evolution is particularly well suited for non-Markov problems and policy structures for which gradients are unnatural or difficult to compute. In addition, the chapter surveys evolutionary neural networks for behavior learning.

PROBABILISTIC MODELS OF SELF AND OTHERS

Current artificial intelligence has become more and more *statistical* and *probabilistic*. Advances in the field of *probabilistic graphical models* are used virtually everywhere, and results for these models – both theoretical as computational – are effectively used in many sub-fields. This is no different for reinforcement learning. There are several large sub-fields in which the use of probabilistic models, such as Bayesian networks, is common practice and the employment of such a universal set of representations and computational techniques enables fruitful connections to other research employing similar models.

The first chapter, **Bayesian Reinforcement Learning** by *Nikos Vlassis, Mohammad Ghavamzadeh, Shie Mannor and Pascal Poupart* surveys Bayesian techniques for reinforcement learning. Learning sequential decision making under uncertainty can be cast in a Bayesian universe where interaction traces provide samples (evidence), and Bayesian inference and learning can be used to find optimal decision strategies in a rigorous probabilistic fashion. The next chapter, **Partially Observable Markov Decision Processes** by *Matthijs Spaan* surveys representations and techniques for partially observable problems which are very often cast in a probabilistic framework such as a dynamic Bayesian network, and where probabilistic inference is needed to infer underlying hidden (unobserved) states. The chapter surveys both model-based as well as model-free methods. Whereas POMDPs are usually modeled in terms of belief states that capture some form of history (or memory), a more recent class of methods that focuses on the *future* is surveyed in the chapter **Predictively Defined Representations of State** by *David Wingate*. These techniques maintain a belief state used for action selection in terms of probabilistic predictions about future events. Several techniques are described in which these predictions are represented compactly and where these are updated based on experience in the world. So far, most methods focus on the prediction (or; evaluation) problem, and less on control. The fourth chapter, **Game Theory and Multi-agent Reinforcement Learning** by *Ann Nowé, Peter Vrancx and Yann-Michaël De Hauwere* moves to a more general set of problems in which multiple agents learn and interact. It surveys game-theoretic and multi-agent approaches in reinforcement learning and shows techniques used to optimize agents in the context of other (learning) agents. The final chapter in this part, **Decentralized POMDPs** by *Frans Oliehoek* surveys

model-based (dynamic programming) techniques for systems consisting of multiple agents that have to cooperatively solve a large task that is decomposed into a set of POMDPs. Such models for example appear in domains where multiple sensors in different locations together may provide essential information on how to act optimally in the world. This chapter builds on methods found in both POMDPs and multi-agent situations.

DOMAINS AND BACKGROUND

As we have said in the beginning of this preface, reinforcement learning appears as a method in many other fields of artificial intelligence, to optimize behaviors. Thus, in addition to the many algorithmic advances as described in the previous three parts of the book, we wanted to include surveys of areas in which reinforcement learning has been applied successfully. This part features chapters on robotics and games. In addition, a third chapter reflects the growing interest in connecting reinforcement learning and cognitive neuroscience.

The first chapter, **Psychological and Neuroscientific Connections with Reinforcement Learning** by *Ashvin Shah* surveys the connection between reinforcement learning methods on the one hand and cognition and neuroscience on the other. Originally many reinforcement learning techniques were derived from insights developed in psychology by for example Skinner, Thorndike and Watson, and still much cross-fertilization between psychology and reinforcement learning can happen. Lately, due to advances in theory about the brain, and especially because testing and measuring of brain activity (fMRI, EEG, etc.) has become much better, much research tries to either 1) explain research findings about the brain in terms of reinforcement learning techniques, i.e. which algorithms do really happen in the brain or 2) get inspired by the inner workings of the brain to come up with new algorithms. The second chapter in this part, **Reinforcement Learning in Games** by *István Szita* surveys the use of reinforcement learning in games. Games is more a general term here than as used in one of the previous chapters on game theory. Indeed, games in this chapter amount to board games such as Backgammon and Checkers, but also computer games such as role-playing and real-time strategy games. Games are often an exciting test bed for reinforcement learning algorithms (see for example also Tetris and Mario in the mentioned reinforcement learning competitions), and in addition to giving many examples, this chapter also tries to outline the main important aspects involved when applying reinforcement learning in game situations. The third chapter in this part, **Reinforcement Learning in Robotics: a Survey** by *Jens Kober and Jan Peters* rigorously describes the application of reinforcement learning to robotics problems. Robotics, because it works with the real, physical world, features many problems that are challenging for the robust application of reinforcement learning. Huge amounts of noisy data, slow training and testing on real robots, the reality gap between simulators and the real world, and scaling up to high-dimensional state spaces are just some of the challenging problems discussed here. Robotics is an exciting area also because of the added possibilities of putting humans in the loop which can create extra opportunities for imitation learning, learning from demonstration, and using humans as teachers for robots.

ACKNOWLEDGEMENTS

Crafting a book such as this can not be done overnight. Many people have put a lot of work in it to make it happen. First of all, we would like to give a big thanks to all the authors who have put in all their expertise, time and creativity to write excellent surveys of their sub-fields. Writing a survey usually takes some extra effort, since it requires that you know much about a topic, but in addition that you can put all relevant works in a more general framework. As editors, we are very happy with the way the authors have accomplished this difficult, yet very useful, task.

A second group of people we would like to thank are the reviewers. They have provided us with very thorough, and especially very constructive, reviews and these have made the book even better. We thank these reviewers who agreed to put their names in the book; thank you very much for all your help: Andrea Bonarini, Prasad Tadepalli, Sarah Ostentoski, Rich Sutton, Daniel Kudenko, Jesse Hoey, Christopher Amato, Damien Ernst, Remi Munos, Johannes Fuernkrantz, Juergen Schmidhuber, Thomas Rückstieß, Joelle Pineau, Dimitri Bertsekas, John Asmuth, Lisa Torrey, Yael Niv, Te Thamrongrattanarit, Michael Littman and Csaba Szepesvári.

Thanks also to Rich Sutton who was so kind to write the foreword to this book. We both consider him as one of the main figures in reinforcement learning, and in all respects we admire him for all the great contributions he has made to the field. He was there in the beginning of modern reinforcement learning, but still he continuously introduces novel, creative new ways to let agents learn. Thanks Rich!

Editing a book such as this is made much more convenient if you can fit it in your daily scientific life. In that respect, Martijn would like to thank both the Katholieke Universiteit Leuven (Belgium) as well as the Radboud University Nijmegen (The Netherlands) for their support. Marco would like to thank the University of Groningen (The Netherlands) for the same kind of support.

Last but not least, we would like to thank you, the reader, to having picked this book and having started to read it. We hope it will be useful to you, and hope that the work you are about to embark on will be incorporated in a subsequent book on reinforcement learning.

Groningen, Nijmegen,
November 2011

Marco Wiering
Martijn van Otterlo

Contents

Part I Introductory Part

1 Reinforcement Learning and Markov Decision Processes	3
<i>Martijn van Otterlo, Marco Wiering</i>	
1.1 Introduction	3
1.2 Learning Sequential Decision Making	5
1.3 A Formal Framework	10
1.3.1 Markov Decision Processes	10
1.3.2 Policies	13
1.3.3 Optimality Criteria and Discounting	13
1.4 Value Functions and Bellman Equations	15
1.5 Solving Markov Decision Processes	17
1.6 Dynamic Programming: Model-Based Solution Techniques	19
1.6.1 Fundamental DP Algorithms	20
1.6.2 Efficient DP Algorithms	24
1.7 Reinforcement Learning: Model-Free Solution Techniques	27
1.7.1 Temporal Difference Learning	29
1.7.2 Monte Carlo Methods	33
1.7.3 Efficient Exploration and Value Updating	34
1.8 Conclusions	39
References	39

Part II Efficient Solution Frameworks

2 Batch Reinforcement Learning	45
<i>Sascha Lange, Thomas Gabel, Martin Riedmiller</i>	
2.1 Introduction	45
2.2 The Batch Reinforcement Learning Problem	46
2.2.1 The Batch Learning Problem	46
2.2.2 The Growing Batch Learning Problem	48
2.3 Foundations of Batch RL Algorithms	49

2.4	Batch RL Algorithms	52
2.4.1	Kernel-Based Approximate Dynamic Programming	53
2.4.2	Fitted Q Iteration	55
2.4.3	Least-Squares Policy Iteration	57
2.4.4	Identifying Batch Algorithms	58
2.5	Theory of Batch RL	60
2.6	Batch RL in Practice.....	61
2.6.1	Neural Fitted Q Iteration (NFQ)	61
2.6.2	NFQ in Control Applications	63
2.6.3	Batch RL for Learning in Multi-agent Systems	65
2.6.4	Deep Fitted Q Iteration	67
2.6.5	Applications/ Further References	69
2.7	Summary	70
	References	71
3	Least-Squares Methods for Policy Iteration.....	75
	<i>Lucian Buşniciu, Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos, Robert Babuška, Bart De Schutter</i>	
3.1	Introduction	76
3.2	Preliminaries: Classical Policy Iteration.....	77
3.3	Least-Squares Methods for Approximate Policy Evaluation	79
3.3.1	Main Principles and Taxonomy	79
3.3.2	The Linear Case and Matrix Form of the Equations	81
3.3.3	Model-Free Implementations	85
3.3.4	Bibliographical Notes	89
3.4	Online Least-Squares Policy Iteration	89
3.5	Example: Car on the Hill	91
3.6	Performance Guarantees	94
3.6.1	Asymptotic Convergence and Guarantees	95
3.6.2	Finite-Sample Guarantees	98
3.7	Further Reading	104
	References	106
4	Learning and Using Models	111
	<i>Todd Hester, Peter Stone</i>	
4.1	Introduction	112
4.2	What Is a Model?	113
4.3	Planning	115
4.3.1	Monte Carlo Methods	115
4.4	Combining Models and Planning	118
4.5	Sample Complexity	120
4.6	Factored Domains	122
4.7	Exploration	126
4.8	Continuous Domains	130
4.9	Empirical Comparisons	133
4.10	Scaling Up	135

4.11	Conclusion	137
References		138
5	Transfer in Reinforcement Learning: A Framework and a Survey . . .	143
	<i>Alessandro Lazaric</i>	
5.1	Introduction	143
5.2	A Framework and a Taxonomy for Transfer in Reinforcement Learning	145
	5.2.1 Transfer Framework	145
	5.2.2 Taxonomy	148
5.3	Methods for Transfer from Source to Target with a Fixed State-Action Space	155
	5.3.1 Problem Formulation	155
	5.3.2 Representation Transfer	156
	5.3.3 Parameter Transfer	158
5.4	Methods for Transfer across Tasks with a Fixed State-Action Space	159
	5.4.1 Problem Formulation	159
	5.4.2 Instance Transfer	160
	5.4.3 Representation Transfer	161
	5.4.4 Parameter Transfer	162
5.5	Methods for Transfer from Source to Target Tasks with a Different State-Action Spaces	164
	5.5.1 Problem Formulation	164
	5.5.2 Instance Transfer	166
	5.5.3 Representation Transfer	166
	5.5.4 Parameter Transfer	167
5.6	Conclusions and Open Questions	168
	References	169
6	Sample Complexity Bounds of Exploration	175
	<i>Lihong Li</i>	
6.1	Introduction	175
6.2	Preliminaries	176
6.3	Formalizing Exploration Efficiency	178
	6.3.1 Sample Complexity of Exploration and PAC-MDP	178
	6.3.2 Regret Minimization	180
	6.3.3 Average Loss	182
	6.3.4 Bayesian Framework	183
6.4	A Generic PAC-MDP Theorem	184
6.5	Model-Based Approaches	186
	6.5.1 Rmax	186
	6.5.2 A Generalization of Rmax	188
6.6	Model-Free Approaches	196
6.7	Concluding Remarks	199
	References	200

Part III Constructive-Representational Directions

7 Reinforcement Learning in Continuous State and Action Spaces	207
<i>Hado van Hasselt</i>	
7.1 Introduction	207
7.1.1 Markov Decision Processes in Continuous Spaces	208
7.1.2 Methodologies to Solve a Continuous MDP	211
7.2 Function Approximation	212
7.2.1 Linear Function Approximation	213
7.2.2 Non-linear Function Approximation	217
7.2.3 Updating Parameters	218
7.3 Approximate Reinforcement Learning	223
7.3.1 Value Approximation	223
7.3.2 Policy Approximation	229
7.4 An Experiment on a Double-Pole Cart Pole	238
7.5 Conclusion	242
References	243
8 Solving Relational and First-Order Logical Markov Decision Processes: A Survey	253
<i>Martijn van Otterlo</i>	
8.1 Introduction to Sequential Decisions in Relational Worlds	253
8.1.1 MDPs: Representation and Generalization	254
8.1.2 Short History and Connections to Other Fields	256
8.2 Extending MDPs with Objects and Relations	257
8.2.1 Relational Representations and Logical Generalization	257
8.2.2 Relational Markov Decision Processes	258
8.2.3 Abstract Problems and Solutions	259
8.3 Model-Based Solution Techniques	261
8.3.1 The Structure of Bellman Backups	262
8.3.2 Exact Model-Based Algorithms	263
8.3.3 Approximate Model-Based Algorithms	266
8.4 Model-Free Solutions	268
8.4.1 Value-Function Learning with Fixed Generalization	269
8.4.2 Value Functions with Adaptive Generalization	270
8.4.3 Policy-Based Solution Techniques	274
8.5 Models, Hierarchies, and Bias	276
8.6 Current Developments	280
8.7 Conclusions and Outlook	283
References	283
9 Hierarchical Approaches	293
<i>Bernhard Hengst</i>	
9.1 Introduction	293
9.2 Background	296
9.2.1 Abstract Actions	297

Contents	XXIII
----------	-------

9.2.2	Semi-Markov Decision Problems	297
9.2.3	Structure	300
9.2.4	State Abstraction	301
9.2.5	Value-Function Decomposition	303
9.2.6	Optimality	303
9.3	Approaches to Hierarchical Reinforcement Learning (HRL)	305
9.3.1	Options	306
9.3.2	HAMQ-Learning	307
9.3.3	MAXQ	309
9.4	Learning Structure	313
9.4.1	HEXQ	315
9.5	Related Work and Ongoing Research	317
9.6	Summary	319
	References	319
10	Evolutionary Computation for Reinforcement Learning	325
	<i>Shimon Whiteson</i>	
10.1	Introduction	325
10.2	Neuroevolution	328
10.3	TWEANNs	330
10.3.1	Challenges	332
10.3.2	NEAT	333
10.4	Hybrids	334
10.4.1	Evolutionary Function Approximation	335
10.4.2	XCS	336
10.5	Coevolution	339
10.5.1	Cooperative Coevolution	339
10.5.2	Competitive Coevolution	342
10.6	Generative and Developmental Systems	343
10.7	On-Line Methods	345
10.7.1	Model-Based Methods	345
10.7.2	On-Line Evolutionary Computation	346
10.8	Conclusion	347
	References	348

Part IV Probabilistic Models of Self and Others

11	Bayesian Reinforcement Learning	359
	<i>Nikos Vlassis, Mohammad Ghavamzadeh, Shie Mannor, Pascal Poupart</i>	
11.1	Introduction	359
11.2	Model-Free Bayesian Reinforcement Learning	361
11.2.1	Value-Function Based Algorithms	361
11.2.2	Policy Gradient Algorithms	365
11.2.3	Actor-Critic Algorithms	369
11.3	Model-Based Bayesian Reinforcement Learning	372
11.3.1	POMDP Formulation of Bayesian RL	372

11.3.2	Bayesian RL via Dynamic Programming	373
11.3.3	Approximate Online Algorithms	376
11.3.4	Bayesian Multi-Task Reinforcement Learning	377
11.3.5	Incorporating Prior Knowledge	379
11.4	Finite Sample Analysis and Complexity Issues	380
11.5	Summary and Discussion	382
	References	382
12	Partially Observable Markov Decision Processes	387
	<i>Matthijs T.J. Spaan</i>	
12.1	Introduction	387
12.2	Decision Making in Partially Observable Environments	389
12.2.1	POMDP Model	389
12.2.2	Continuous and Structured Representations	391
12.2.3	Memory for Optimal Decision Making	391
12.2.4	Policies and Value Functions	394
12.3	Model-Based Techniques	395
12.3.1	Heuristics Based on MDP Solutions	396
12.3.2	Value Iteration for POMDPs	397
12.3.3	Exact Value Iteration	400
12.3.4	Point-Based Value Iteration Methods	401
12.3.5	Other Approximate Methods	403
12.4	Decision Making Without a-Priori Models	404
12.4.1	Memoryless Techniques	405
12.4.2	Learning Internal Memory	405
12.5	Recent Trends	408
	References	409
13	Predictively Defined Representations of State	415
	<i>David Wingate</i>	
13.1	Introduction	416
13.1.1	What Is “State”?	416
13.1.2	Which Representation of State?	418
13.1.3	Why Predictions about the Future?	419
13.2	PSRs	420
13.2.1	Histories and Tests	421
13.2.2	Prediction of a Test	422
13.2.3	The System Dynamics Vector	422
13.2.4	The System Dynamics Matrix	423
13.2.5	Sufficient Statistics	424
13.2.6	State	424
13.2.7	State Update	425
13.2.8	Linear PSRs	425
13.2.9	Relating Linear PSRs to POMDPs	426
13.2.10	Theoretical Results on Linear PSRs	427
13.3	Learning a PSR Model	428

13.3.1	The Discovery Problem	428
13.3.2	The Learning Problem	429
13.3.3	Estimating the System Dynamics Matrix	429
13.4	Planning with PSRs	429
13.5	Extensions of PSRs	431
13.6	Other Models with Predictively Defined State	432
13.6.1	Observable Operator Models	433
13.6.2	The Predictive Linear-Gaussian Model	433
13.6.3	Temporal-Difference Networks	434
13.6.4	Diversity Automaton	435
13.6.5	The Exponential Family PSR	435
13.6.6	Transformed PSRs	436
13.7	Conclusion	436
	References	437
14	Game Theory and Multi-agent Reinforcement Learning	441
	<i>Ann Nowé, Peter Vrancx, Yann-Michaël De Hauwere</i>	
14.1	Introduction	441
14.2	Repeated Games	445
14.2.1	Game Theory	445
14.2.2	Reinforcement Learning in Repeated Games	449
14.3	Sequential Games	454
14.3.1	Markov Games	455
14.3.2	Reinforcement Learning in Markov Games	456
14.4	Sparse Interactions in Multi-agent System	461
14.4.1	Learning on Multiple Levels	461
14.4.2	Learning to Coordinate with Sparse Interactions	462
14.5	Further Reading	467
	References	467
15	Decentralized POMDPs	471
	<i>Frans A. Oliehoek</i>	
15.1	Introduction	471
15.2	The Decentralized POMDP Framework	473
15.3	Histories and Policies	475
15.3.1	Histories	475
15.3.2	Policies	476
15.3.3	Structure in Policies	477
15.3.4	The Quality of Joint Policies	479
15.4	Solution of Finite-Horizon Dec-POMDPs	480
15.4.1	Brute Force Search and Dec-POMDP Complexity	480
15.4.2	Alternating Maximization	481
15.4.3	Optimal Value Functions for Dec-POMDPs	481
15.4.4	Forward Approach: Heuristic Search	485
15.4.5	Backwards Approach: Dynamic Programming	489
15.4.6	Other Finite-Horizon Methods	493

15.5	Further Topics	493
15.5.1	Generalization and Special Cases	493
15.5.2	Infinite-Horizon Dec-POMDPs	495
15.5.3	Reinforcement Learning	496
15.5.4	Communication	497
	References	498

Part V Domains and Background

16	Psychological and Neuroscientific Connections with Reinforcement Learning	507
	<i>Ashvin Shah</i>	
16.1	Introduction	507
16.2	Classical (or Pavlovian) Conditioning	508
16.2.1	Behavior	509
16.2.2	Theory	511
16.2.3	Summary and Additional Considerations	512
16.3	Operant (or Instrumental) Conditioning	513
16.3.1	Behavior	513
16.3.2	Theory	514
16.3.3	Model-Based Versus Model-Free Control	516
16.3.4	Summary and Additional Considerations	517
16.4	Dopamine	518
16.4.1	Dopamine as a Reward Prediction Error	518
16.4.2	Dopamine as a General Reinforcement Signal	520
16.4.3	Summary and Additional Considerations	521
16.5	The Basal Ganglia	521
16.5.1	Overview of the Basal Ganglia	522
16.5.2	Neural Activity in the Striatum	523
16.5.3	Cortico-basal Ganglia-thalamic Loops	524
16.5.4	Summary and Additional Considerations	526
16.6	Chapter Summary	527
	References	528
17	Reinforcement Learning in Games	539
	<i>István Szita</i>	
17.1	Introduction	539
17.1.1	Aims and Structure	540
17.1.2	Scope	541
17.2	A Showcase of Games	541
17.2.1	Backgammon	542
17.2.2	Chess	545
17.2.3	Go	550
17.2.4	Tetris	555
17.2.5	Real-Time Strategy Games	558
17.3	Challenges of Applying Reinforcement Learning to Games	561

17.3.1	Representation Design	561
17.3.2	Exploration	564
17.3.3	Source of Training Data	565
17.3.4	Dealing with Missing Information	566
17.3.5	Opponent Modelling	567
17.4	Using RL in Games	568
17.4.1	Opponents That Maximize Fun	568
17.4.2	Development-Time Learning	570
17.5	Closing Remarks	571
	References	572
18	Reinforcement Learning in Robotics: A Survey	579
	<i>Jens Kober, Jan Peters</i>	
18.1	Introduction	579
18.2	Challenges in Robot Reinforcement Learning	581
18.2.1	Curse of Dimensionality	582
18.2.2	Curse of Real-World Samples	583
18.2.3	Curse of Real-World Interactions	584
18.2.4	Curse of Model Errors	584
18.2.5	Curse of Goal Specification	585
18.3	Foundations of Robot Reinforcement Learning	585
18.3.1	Value Function Approaches	586
18.3.2	Policy Search	588
18.4	Tractability through Representation	589
18.4.1	Smart State-Action Discretization	590
18.4.2	Function Approximation	592
18.4.3	Pre-structured Policies	592
18.5	Tractability through Prior Knowledge	594
18.5.1	Prior Knowledge through Demonstrations	594
18.5.2	Prior Knowledge through Task Structuring	596
18.5.3	Directing Exploration with Prior Knowledge	596
18.6	Tractability through Simulation	596
18.6.1	Role of Models	597
18.6.2	Mental Rehearsal	598
18.6.3	Direct Transfer from Simulated to Real Robots	599
18.7	A Case Study: Ball-in-a-Cup	599
18.7.1	Experimental Setting: Task and Reward	599
18.7.2	Appropriate Policy Representation	601
18.7.3	Generating a Teacher's Demonstration	601
18.7.4	Reinforcement Learning by Policy Search	601
18.7.5	Use of Simulations in Robot Reinforcement Learning	603
18.7.6	Alternative Approach with Value Function Methods	603
18.8	Conclusion	603
	References	604

Part VI Closing

19 Conclusions, Future Directions and Outlook	613
<i>Marco Wiering, Martijn van Otterlo</i>	
19.1 Looking Back	613
19.1.1 What Has Been Accomplished?.....	613
19.1.2 Which Topics Were Not Included?	614
19.2 Looking into the Future	620
19.2.1 Things That Are Not Yet Known	620
19.2.2 Seemingly Impossible Applications for RL	622
19.2.3 Interesting Directions	623
19.2.4 Experts on Future Developments	624
References	626
Index	631

List of Contributors

Robert Babuška

Delft Center for Systems and Control, Delft University of Technology,
The Netherlands
e-mail: r.babuska@tudelft.nl

Lucian Buşoniu

Team SequeL, INRIA Lille-Nord Europe, France
e-mail: ion-lucian.busoniu@inria.fr

Thomas Gabel

Albert-Ludwigs-Universität, Faculty of Engineering, Germany,
e-mail: tgabel@informatik.uni-freiburg.de

Mohammad Ghavamzadeh

Team SequeL, INRIA Lille-Nord Europe, France
e-mail: mohammad.ghavamzadeh@inria.fr

Hado van Hasselt

Centrum Wiskunde en Informatica (CWI, Center for Mathematics and Computer
Science), Amsterdam, The Netherlands
e-mail: H.van.Hasselt@cwi.nl

Yann-Michaël De Hauwere

Vrije Universiteit Brussel, Belgium
e-mail: ydehauwe@vub.ac.be

Bernhard Hengst

School of Computer Science and Engineering,
University of New South Wales, Sydney, Australia
e-mail: bernhardh@cse.unsw.edu.au

Todd Hester

Department of Computer Science, The University of Texas at Austin, USA
e-mail: todd@cs.utexas.edu

Jens Kober

1) Intelligent Autonomous Systems Institute, Technische Universitaet Darmstadt, Darmstadt, Germany; 2) Robot Learning Lab, Max-Planck Institute for Intelligent Systems, Tübingen, Germany
e-mail: jens.kober@tuebingen.mpg.de

Sascha Lange

Albert-Ludwigs-Universität Freiburg, Faculty of Engineering, Germany
e-mail: slange@informatik.uni-freiburg.de

Alessandro Lazaric

Team SequeL, INRIA Lille-Nord Europe, France
e-mail: alessandro.lazaric@inria.fr

Lihong Li

Yahoo! Research, Santa Clara, USA
e-mail: lihong@yahoo-inc.com

Shie Mannor

Technion, Haifa, Israel
e-mail: shie@ee.technion.ac.il

Rémi Munos

Team SequeL, INRIA Lille-Nord Europe, France
e-mail: remi.munos@inria.fr

Frans Oliehoek

CSAIL, Massachusetts Institute of Technology
e-mail: fao@csail.mit.edu

Ann Nowé

Vrije Universiteit Brussel, Belgium
e-mail: anowe@vub.ac.be

Martijn van Otterlo

Radboud University Nijmegen, The Netherlands
e-mail: m.vanotterlo@donders.ru.nl

Jan Peters

1) Intelligent Autonomous Systems Institute, Technische Universitaet Darmstadt, Darmstadt, Germany; 2) Robot Learning Lab, Max-Planck Institute for Intelligent Systems, Tübingen, Germany
e-mail: jan.peters@tuebingen.mpg.de

Pascal Poupart

University of Waterloo, Canada
e-mail: ppoupart@cs.uwaterloo.ca

Martin Riedmiller

Albert-Ludwigs-Universität Freiburg, Faculty of Engineering, Germany
e-mail: riedmiller@informatik.uni-freiburg.de

Bart De Schutter

Delft Center for Systems and Control,
Delft University of Technology, The Netherlands
e-mail: b.deschutter@tudelft.nl

Ashvin Shah

Department of Psychology, University of Sheffield, Sheffield, UK
e-mail: ashvin@gmail.com

Matthijs Spaan

Institute for Systems and Robotics, Instituto Superior Técnico, Lisbon, Portugal
e-mail: mtjspa@isr.ist.utl.pt

Peter Stone

Department of Computer Science, The University of Texas at Austin, USA
e-mail: pstone@cs.utexas.edu

István Szita

University of Alberta, Canada
e-mail: szityu@gmail.com

Nikos Vlassis

(1) Luxembourg Centre for Systems Biomedicine, University of Luxembourg,
and (2) OneTree Financials, Luxembourg
e-mail: nikos.vlassis@uni.lu, nikos@onetreefinancials.com

Peter Vranckx

Vrije Universiteit Brussel, Belgium
e-mail: pvrancx@vub.ac.be

Shimon Whiteson

Informatics Institute, University of Amsterdam, The Netherlands
e-mail: s.a.whiteson@uva.nl

Marco Wiering

Department of Artificial Intelligence, University of Groningen, The Netherlands
e-mail: m.a.wiering@rug.nl

David Wingate

Massachusetts Institute of Technology, Cambridge, USA
e-mail: wingated@mit.edu

Acronyms

AC	Actor-Critic
AO	Action-Outcome
BAC	Bayesian Actor-Critic
BEETLE	Bayesian Exploration-Exploitation Tradeoff in Learning
BG	Basal Ganglia
BQ	Bayesian Quadrature
BQL	Bayesian Q-learning
BPG	Bayesian Policy Gradient
BRM	Bellman Residual Minimization (generic; BRM-Q for Q-functions; BRM-V for V-functions)
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CPPN	Compositional Pattern Producing Network
CoSyNE	Cooperative Synapse Coevolution
CR	Conditioned Response
CS	Conditioned Stimulus
DA	Dopamine
DBN	Dynamic Bayesian Network
DEC-MDP	Decentralized Markov Decision Process
DFQ	Deep Fitted Q iteration
DP	Dirichlet process
DP	Dynamic Programming
DTR	Decision-Theoretic Regression
EDA	Estimation of Distribution Algorithm
ESP	Enforced SubPopulations
FODTR	First-Order (Logical) Decision-Theoretic Regression
FQI	Fitted Q Iteration
GP	Gaussian Process
GPI	Generalized Policy Iteration
GPTD	Gaussian Process Temporal Difference
HBM	Hierarchical Bayesian model
HRL	Hierarchical Reinforcement Learning

ILP	Inductive Logic Programming
KADP	Kernel-based Approximate Dynamic Programming
KR	Knowledge Representation
KWIK	Knows What It Knows
LCS	Learning Classifier System
LSPE	Least-Squares Policy Evaluation (generic; ; LSPE-Q for Q-functions; LSPE-V for V-functions)
LSPI	Least-Squares Policy Iteration
LSTDQ	Least-Squares Temporal Difference Q-Learning
LSTD	Least-Squares Temporal Difference (generic; LSTD-Q for Q-functions; LSTD-V for V-functions)
MB	Mistake Bound
MC	Monte-Carlo
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
ML	Machine Learning
MTL	Multi-Task Learning
MTRL	Multi-Task Reinforcement Learning
NEAT	NeuroEvolution of Augmenting Topologies
NFQ	Neural Fitted Q iteration
PAC	Probably Approximately Correct
PAC-MDP	Probably Approximately Correct in Markov Decision Process
PMBGA	Probabilistic Model-Building Genetic Algorithm
PI	Policy Iteration
PIAGeT	Policy Iteration using Abstraction and Generalization Techniques
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
RMDP	Relational Markov Decision Process
SANE	Symbiotic Adaptive NeuroEvolution
sGA	Structured Genetic Algorithm
SMDP	Semi-Markov Decision Process
SR	Stimulus-Response
SRL	Statistical Relational Learning
TD	Temporal Difference
TWEANN	Topology- and Weight-Evolving Artificial Neural Network
UR	Unconditioned Response
US	Unconditioned Stimulus
VI	Value Iteration
VPI	Value of Perfect Information

Part I

Introductory Part

Chapter 1

Reinforcement Learning and Markov Decision Processes

Martijn van Otterlo and Marco Wiering

Abstract. Situated in between supervised learning and unsupervised learning, the paradigm of reinforcement learning deals with learning in sequential decision making problems in which there is limited feedback. This text introduces the intuitions and concepts behind Markov decision processes and two classes of algorithms for computing optimal behaviors: reinforcement learning and dynamic programming. First the formal framework of Markov decision process is defined, accompanied by the definition of value functions and policies. The main part of this text deals with introducing foundational classes of algorithms for learning optimal behaviors, based on various definitions of optimality with respect to the goal of learning sequential decisions. Additionally, it surveys efficient extensions of the foundational algorithms, differing mainly in the way feedback given by the environment is used to speed up learning, and in the way they concentrate on relevant parts of the problem. For both model-based and model-free settings these efficient extensions have shown useful in scaling up to larger problems.

1.1 Introduction

Markov Decision Processes (MDP) Puterman (1994) are an intuitive and fundamental formalism for *decision-theoretic planning* (DTP) Boutilier et al (1999); Boutilier (1999), reinforcement learning (RL) Bertsekas and Tsitsiklis (1996); Sutton and Barto (1998); Kaelbling et al (1996) and other learning problems in stochastic domains. In this model, an environment is modelled as a set of states and actions can

Martijn van Otterlo
Radboud University Nijmegen, The Netherlands
e-mail: m.vanotterlo@donders.ru.nl

Marco Wiering
Department of Artificial Intelligence, University of Groningen, The Netherlands
e-mail: m.a.wiering@rug.nl

be performed to control the system's state. The goal is to control the system in such a way that some performance criterium is maximized. Many problems such as (stochastic) planning problems, learning robot control and game playing problems have successfully been modelled in terms of an MDP. In fact MDPs have become the *de facto* standard formalism for learning sequential decision making.

DTP Boutilier et al (1999), e.g. planning using decision-theoretic notions to represent uncertainty and plan quality, is an important extension of the AI *planning paradigm*, adding the ability to deal with *uncertainty* in action effects and the ability to deal with less-defined *goals*. Furthermore it adds a significant dimension in that it considers situations in which factors such as resource consumption and uncertainty demand solutions of *varying quality*, for example in *real-time* decision situations. There are many connections between AI planning, research done in the field of *operations research* Winston (1991) and *control theory* Bertsekas (1995), as most work in these fields on *sequential decision making* can be viewed as instances of MDPs. The notion of a *plan* in AI planning, i.e. a series of actions from a start state to a goal state, is extended to the notion of a *policy*, which is mapping from *all* states to an (optimal) action, based on decision-theoretic measures of *optimality* with respect to some goal to be optimized.

As an example, consider a typical planning domain, involving boxes to be moved around and where the goal is to move some particular boxes to a designated area. This type of problems can be solved using AI planning techniques. Consider now a slightly more realistic extension in which some of the actions can fail, or have uncertain side-effects that can depend on factors beyond the operator's control, and where the goal is specified by giving credit for how many boxes are put on the right place. In this type of environment, the notion of a plan is less suitable, because a sequence of actions can have many different outcomes, depending on the effects of the operators used in the plan. Instead, the methods in this chapter are concerned about *policies* that map states onto actions in such a way that the *expected* outcome of the operators will have the intended effects. The expectation over actions is based on a decision-theoretic expectation with respect to their probabilistic outcomes and credits associated with the problem goals. The MDP framework allows for online solutions that *learn* optimal policies gradually through *simulated trials*, and additionally, it allows for *approximated* solutions with respect to resources such as computation time. Finally, the model allows for numeric, decision-theoretic measurement of the *quality* of policies and learning *performance*. For example, policies can be ordered by how much credit they receive, or by how much computation is needed for a particular performance.

This chapter will cover the broad spectrum of methods that have been developed in the literature to compute good or optimal policies for problems modelled as an MDP. The term RL is associated with the more difficult setting in which no (prior) knowledge about the MDP is presented. The task then of the algorithm is to *interact*, or *experiment* with the environment (i.e. the MDP), in order to gain knowledge about how to optimize its behavior, being guided by the evaluative feedback (rewards). The model-based setting, in which the full transition dynamics and reward distributions are known, is usually characterized by the use of *dynamic*

environment	You are in state 65. You have 4 possible actions.
agent	I'll take action 2.
environment	You have received a reward of 7 units. You are now in state 15.
agent	You have 2 possible actions.
environment	I'll take action 1.
agent	You have received a reward of -4 units. You are now in state 65.
environment	You have 4 possible actions.
agent	I'll take action 2.
environment	You have received a reward of 5 units. You are now in state 44.
...	You have 5 possible actions.

Fig. 1.1 Example of interaction between an agent and its environment, from an RL perspective

programming (DP) techniques. However, we will see that the underlying basis is very similar, and that mixed forms occur.

1.2 Learning Sequential Decision Making

RL is a general class of algorithms in the field of machine learning that aims at allowing an *agent* to learn how to behave in an environment, where the only feedback consists of a *scalar* reward signal. RL should not be seen as characterized by a particular class of learning methods, but rather as a learning *problem* or a *paradigm*. The *goal* of the agent is to perform *actions* that maximize the reward signal in the long run.

The distinction between the *agent* and the *environment* might not always be the most intuitive one. We will draw a boundary based on control Sutton and Barto (1998). Everything the agent cannot control is considered part of the environment. For example, although the motors of a robot agent might be considered part of the agent, the exact functioning of them in the environment is beyond the agent's control. It can give commands to gear up or down, but their physical realization can be influenced by many things.

An example of interaction with the environment is given in Figure 1.1. It shows how the interaction between an *agent* and the *environment* can take place. The agent can choose an action in each state, and the *perceptions* the agent gets from the environment are the environment's state after each action plus the scalar reward signal at each step. Here a discrete model is used in which there are distinct numbers for each state and action. The way the interaction is depicted is highly general in the sense that one just talks about states and actions as discrete *symbols*. In the rest of this book we will be more concerned about interactions in which states and actions have more *structure*, such that a state can be something like *there are two blue*

boxes and one white one and you are standing next to a blue box. However, this figure clearly shows the *mechanism* of sequential decision making.

There are several important aspects in learning sequential decision making which we will describe in this section, after which we will describe formalizations in the next sections.

Approaching Sequential Decision Making

There are several classes of algorithms that deal with the problem of sequential decision making. In this book we deal specifically with the topic of *learning*, but some other options exist.

The first solution is the *programming* solution. An intelligent system for sequential decision making can – in principle – be *programmed* to handle all situations. For each possible state an appropriate or optimal action can be specified *a priori*. However, this puts a heavy burden on the designer or programmer of the system. All situations should be foreseen in the design phase and programmed into the agent. This is a tedious and almost impossible task for most interesting problems, and it only works for problems which can be modelled completely. In most realistic problems this is not possible due to the sheer size of the problem, or the intrinsic *uncertainty* in the system. A simple example is *robot control* in which factors such as lighting or temperature can have a large, and unforeseen, influence on the behavior of camera and motor systems. Furthermore, in situations where the problem changes, for example due to new elements in the description of the problem or changing dynamics of the system, a programmed solution will no longer work. Programmed solutions are *brittle* in that they will only work for completely known, static problems with fixed probability distributions.

A second solution uses *search* and *planning* for sequential decision making. The successful chess program *Deep Blue* Schaeffer and Plaat (1997) was able to defeat the human world champion Gary Kasparov by smart, brute force search algorithms that used a model of the dynamics of chess, tuned to Kasparov's style of playing. When the dynamics of the system are known, one can *search* or *plan* from the current state to a desirable goal state. However, when there is uncertainty about the action outcomes standard search and planning algorithms do not apply. *Admissible heuristics* can solve some problems concerning the reward-based nature of sequential decision making, but the probabilistic effects of actions pose a difficult problem. Probabilistic planning algorithms exist, e.g. Kushmerick et al (1995), but their performance is not as good as their deterministic counterparts. An additional problem is that planning and search focus on specific start and goal states. In contrast, we are looking for *policies* which are defined for all states, and are defined with respect to *rewards*.

The third solution is *learning*, and this will be the main topic of this book. Learning has several advantages in sequential decision making. First, it relieves the designer of the system from the difficult task of deciding upon everything in the design phase. Second, it can cope with uncertainty, goals specified in terms of reward

measures, and with changing situations. Third, it is aimed at solving the problem for every state, as opposed to a mere plan from one state to another. Additionally, although a model of the environment can be used or learned, it is not necessary in order to compute optimal policies, such as is exemplified by RL methods. Everything can be learned from interaction with the environment.

Online versus Off-line Learning

One important aspect in the learning task we consider in this book is the distinction between *online* and *off-line* learning. The difference between these two types is influenced by factors such as whether one wants to control a *real-world* entity – such as a robot playing robot soccer or a machine in a factory – or whether all necessary information is available. Online learning performs learning directly on the problem instance. Off-line learning uses a *simulator* of the environment as a cheap way to get many training examples for *safe* and *fast* learning.

Learning the controller directly on the real task is often not possible. For example, the learning algorithms in this chapter sometimes need millions of training instances which can be too time-consuming to collect. Instead, a simulator is much faster, and in addition it can be used to provide arbitrary training situations, including situations that rarely happen in the real system. Furthermore, it provides a "safe" training situation in which the agent can explore and make mistakes. Obtaining negative feedback in the real task in order to learn to avoid these situations, might entail destroying the machine that is controlled, which is unacceptable. Often one uses a simulation to obtain a reasonable policy for a given problem, after which some parts of the behavior are *fine-tuned* on the real task. For example, a simulation might provide the means for learning a reasonable robot controller, but some *physical* factors concerning variance in motor and perception systems of the robot might make additional fine-tuning necessary. A simulation is just a *model* of the real problem, such that small differences between the two are natural, and learning might make up for that difference. Many problems in the literature however, *are* simulations of games and optimization problems, such that the distinction disappears.

Credit Assignment

An important aspect of sequential decision making is the fact that deciding whether an action is "good" or "bad" cannot be decided upon right away. The appropriateness of actions is completely determined by the *goal* the agent is trying to pursue. The real problem is that the effect of actions with respect to the goal can be much *delayed*. For example, the opening moves in chess have a large influence on winning the game. However, between the first opening moves and receiving a reward for winning the game, a couple of tens of moves might have been played. Deciding how to give *credit* to the first moves – which did not get the immediate reward for winning – is a difficult problem called the *temporal credit assignment* problem.

Each move in a winning chess game contributes more or less to the success of the last move, although some moves along this path can be less optimal or even bad. A related problem is the *structural credit assignment* problem, in which the problem is to distribute feedback over the *structure* representing the agent's policy. For example, the policy can be represented by a structure containing parameters (e.g. a neural network). Deciding which parameters have to be updated forms the structural credit assignment problem.

The Exploration-Exploitation Trade-off

If we know a complete model of dynamics of the problem, there exist methods (e.g. DP) that can compute optimal policies from this model. However, in the more general case where we do not have access to this knowledge (e.g. RL), it becomes necessary to *interact* with the environment to learn by *trial-and-error* a correct policy. The agent has to *explore* the environment by performing actions and perceiving their consequences (i.e. the effects on the environments and the obtained rewards). The only feedback the agent gets are rewards, but it does not get information about what is the right action. At some point in time, it will have a policy with a particular performance. In order to see whether there are possible improvements to this policy, it sometimes has to *try out* various actions to see their results. This might result in worse performance because the actions might also be less good than the current policy. However, without trying them, it might never find possible improvements. In addition, if the world is not stationary, the agent has to explore to keep its policy up-to-date. So, in order to *learn* it has to *explore*, but in order to *perform well* it should *exploit* what it already knows. Balancing these two things is called the *exploration-exploitation problem*.

Feedback, Goals and Performance

Compared to supervised learning, the amount of feedback the learning system gets in RL, is much less. In supervised learning, for every learning sample the correct output is given in a training set. The performance of the learning system can be measured relative to the number of correct answers, resulting in a *predictive accuracy*. The difficulty lies in learning this mapping, and whether this mapping *generalizes* to new, unclassified, examples. In unsupervised learning, the difficulty lies in constructing a useful partitioning of the data such that classes naturally arise. In reinforcement learning there is only some information available about performance, in the form of one *scalar* signal. This feedback system is *evaluative* rather than being *instructive*. Using this limited signal for feedback renders a need to put more effort in using it to evaluate and improve behavior during learning.

A second aspect about feedback and performance is related to the stochastic nature of the problem formulation. In supervised and unsupervised learning, the data is usually considered *static*, i.e. a data set is given and performance can be measured

with respect to this data. The learning samples for the learner originate from a fixed distribution, i.e. the data set. From an RL perspective, the data can be seen as a *moving target*. The learning process is driven by the current policy, but this policy will change over time. That means that the *distribution* over states and rewards will change because of this. In machine learning the problem of a changing distribution of learning samples is termed *concept drift* Maloof (2003) and it demands special features to deal with it. In RL this problem is dealt with by exploration, a constant interaction between evaluation and improvement of policies and additionally the use of *learning rate adaption schemes*.

A third aspect of feedback is the question "*where do the numbers come from?*". In many sequential decision tasks, suitable reward functions present themselves quite naturally. For games in which there are winning, losing and draw situations, the reward function is easy to specify. In some situations special care has to be taken in giving rewards for states or actions, and also their *relative size* is important. When the agent will encounter a large negative reward before it finally gets a small positive reward, this positive reward might get *overshadowed*. All problems posed will have *some* optimal policy, but it depends on whether the reward function is in accordance with the right goals, whether the policy will tackle the *right* problem. In some problems it can be useful to provide the agent with rewards for reaching intermediate *subgoals*. This can be helpful in problems which require very long action sequences.

Representations

One of the most important aspects in learning sequential decision making is *representation*. Two central issues are *what* should be represented, and *how* things should be represented. The first issue is dealt with in this chapter. Key components that can or should be represented are models of the dynamics of the environment, reward distributions, value functions and policies. For some algorithms all components are explicitly stored in tables, for example in classic DP algorithms. Actor-critic methods keep separate, explicit representations of both value functions and policies. However, in most RL algorithms just a value function is represented whereas policy decisions are derived from this value function online. Methods that search in policy space do not represent value functions explicitly, but instead an explicitly represented policy is used to compute values when necessary. Overall, the choice for *not* representing certain elements can influence the choice for a type of algorithm, and its efficiency.

The question of *how* various structures can be represented is dealt with extensively in this book, starting from the next chapter. Structures such as policies, transition functions and value functions can be represented in more compact form by using various *structured* knowledge representation formalisms and this enables much more efficient solution mechanisms and scaling up to larger domains.

1.3 A Formal Framework

The elements of the RL problem as described in the introduction to this chapter can be formalized using the *Markov decision process* (MDP) framework. In this section we will formally describe components such as *states* and *actions* and *policies*, as well as the *goals* of learning using different kinds of *optimality criteria*. MDPs are extensively described in Puterman (1994) and Boutilier et al (1999). They can be seen as stochastic extensions of finite automata and also as *Markov processes* augmented with actions and rewards.

Although general MDPs may have infinite (even uncountable) state and action spaces, we limit the discussion to finite-state and finite-action problems. In the next chapter we will encounter continuous spaces and in later chapters we will encounter situations arising in the first-order logic setting in which infinite spaces can quite naturally occur.

1.3.1 *Markov Decision Processes*

MDPs consist of states, actions, transitions between states and a reward function definition. We consider each of them in turn.

States

The set of environmental states S is defined as the finite set $\{s^1, \dots, s^N\}$ where the *size* of the state space is N , i.e. $|S| = N$. A *state* is a unique characterization of all that is important in a state of the problem that is modelled. For example, in chess a complete configuration of board pieces of both black and white, is a state. In the next chapter we will encounter the use of *features* that *describe* the state. In those contexts, it becomes necessary to distinguish between *legal* and *illegal* states, for some combinations of features might not result in an actually existing state in the problem. In this chapter, we will confine ourselves to the discrete state set S in which each state is represented by a *distinct symbol*, and all states $s \in S$ are legal.

Actions

The set of actions A is defined as the finite set $\{a^1, \dots, a^K\}$ where the *size* of the action space is K , i.e. $|A| = K$. Actions can be used to *control* the system state. The set of actions that can be applied in some particular state $s \in S$, is denoted $A(s)$, where $A(s) \subseteq A$. In some systems, not all actions can be applied in every state, but in general we will assume that $A(s) = A$ for all $s \in S$. In more structured representations (e.g. by means of *features*), the fact that some actions are not applicable in some

states, is modelled by a *precondition function* $\text{pre} : S \times A \rightarrow \{\text{true}, \text{false}\}$, stating whether action $a \in A$ is applicable in state $s \in S$.

The Transition Function

By applying action $a \in A$ in a state $s \in S$, the system makes a *transition* from s to a new state $s' \in S$, based on a probability distribution over the set of possible transitions. The transition function T is defined as $T : S \times A \times S \rightarrow [0,1]$, i.e. the probability of ending up in state s' after doing action a in state s is denoted $T(s,a,s')$. It is required that for all actions a , and all states s and s' , $T(s,a,s') \geq 0$ and $T(s,a,s') \leq 1$. Furthermore, for all states s and actions a , $\sum_{s' \in S} T(s,a,s') = 1$, i.e. T defines a *proper probability distribution over possible next states*. Instead of a precondition function, it is also possible to set¹ $T(s,a,s') = 0$ for all states $s' \in S$ if a is not applicable in s . For talking about the *order* in which actions occur, we will define a discrete *global clock*, $t = 1, 2, \dots$. Using this, the notation s_t denotes the state at time t and s_{t+1} denotes the state at time $t + 1$. This enables to compare different states (and actions) occurring ordered in time during interaction.

The system being controlled is *Markovian* if the result of an action does not depend on the previous actions and visited states (history), but only depends on the current state, i.e.

$$P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots) = P(s_{t+1} | s_t, a_t) = T(s_t, a_t, s_{t+1})$$

The idea of Markovian dynamics is that the current state s gives enough information to make an optimal decision; it is not important which states and actions preceded s . Another way of saying this, is that if you select an action a , the probability distribution over next states is the same as the last time you tried this action in the same state. More general models can be characterized by being *k-Markov*, i.e. the last k states are sufficient, such that *Markov* is actually *1-Markov*. Though, each *k-Markov* problem can be transformed into an equivalent *Markov* problem. The *Markov property* forms a boundary between the MDP and more general models such as POMDPs.

The Reward Function

The *reward function*² specifies rewards for being in a state, or doing some action in a state. The *state reward* function is defined as $R : S \rightarrow \mathbb{R}$, and it specifies the

¹ Although this is the same, the explicit distinction between an action not being applicable in a state and a zero probability for transitions with that action, is lost in this way.

² Although we talk about *rewards* here, with the usual connotation of something positive, the reward function merely gives a *scalar* feedback signal. This can be interpreted as negative (*punishment*) or positive (*reward*). The various origins of work in MDPs in the literature creates an additional confusion with the reward function. In the *operations research* literature, one usually speaks of a *cost function* instead and the goal of learning and optimization is to *minimize* this function.

reward obtained in states. However, two other definitions exist. One can define either $R : S \times A \rightarrow \mathbb{R}$ or $R : S \times A \times S \rightarrow \mathbb{R}$. The first one gives rewards for performing an action in a state, and the second gives rewards for particular transitions between states. All definitions are interchangeable though the last one is convenient in *model-free* algorithms (see Section 1.7), because there we usually need both the starting state and the resulting state in backing up values. Throughout this book we will mainly use $R(s,a,s')$, but deviate from this when more convenient.

The reward function is an important part of the MDP that specifies implicitly the *goal* of learning. For example, in episodic tasks such as in the games *Tic-Tac-Toe* and chess, one can assign all states in which the agent has won a positive reward value, all states in which the agent loses a negative reward value and a zero reward value in all states where the final outcome of the game is a draw. The goal of the agent is to reach positive valued states, which means winning the game. Thus, the reward function is used to give *direction* in which way the system, i.e. the MDP, should be controlled. Often, the reward function assigns non-zero reward to non-goal states as well, which can be interpreted as defining *sub-goals* for learning.

The Markov Decision Process

Putting all elements together results in the definition of a *Markov decision process*, which will be the base model for the large majority of methods described in this book.

Definition 1.3.1. A *Markov decision process* is a tuple $\langle S, A, T, R \rangle$ in which S is a finite set of states, A a finite set of actions, T a transition function defined as $T : S \times A \times S \rightarrow [0,1]$ and R a reward function defined as $R : S \times A \times S \rightarrow \mathbb{R}$.

The transition function T and the reward function R together define the *model* of the MDP. Often MDPs are depicted as a state transition graph where the nodes correspond to states and (directed) edges denote transitions. A typical domain that is frequently used in the MDP literature is the *maze* Matthews (1922), in which the reward function assigns a positive reward for reaching the exit state.

There are several distinct types of systems that can be modelled by this definition of an MDP. In *episodic tasks*, there is the notion of *episodes* of some length, where the goal is to take the agent from a starting state to a *goal state*. An *initial state distribution* $I : S \rightarrow [0,1]$ gives for each state the probability of the system being started in that state. Starting from a state s the system progresses through a sequence of states, based on the actions performed. In episodic tasks, there is a specific subset $G \subseteq S$, denoted *goal state area* containing states (usually with some distinct reward) where the process *ends*. We can furthermore distinguish between *finite, fixed horizon* tasks in which each episode consists of a fixed number of steps, *indefinite horizon* tasks in which each episode can end but episodes can have arbitrary length, and *infinite horizon* tasks where the system does not end at all. The last type of model is usually called a *continuing task*.

Episodic tasks, i.e. in which there are so-called *goal states*, can be modelled using the same model defined in Definition 1.3.1. This is usually modelled by means of *absorbing states* or *terminal states*, e.g. states from which every action results in a transition to that same state with probability 1 and reward 0. Formally, for an absorbing state s , it holds that $T(s,a,s) = 1$ and $R(s,a,s') = 0$ for all states $s' \in S$ and actions $a \in A$. When entering an absorbing state, the process is reset and restarts in a new starting state. Episodic tasks and absorbing states can in this way be elegantly modelled in the same framework as continuing tasks.

1.3.2 Policies

Given an MDP $\langle S, A, T, R \rangle$, a policy is a computable function that outputs for each state $s \in S$ an action $a \in A$ (or $a \in A(s)$). Formally, a *deterministic* policy π is a function defined as $\pi : S \rightarrow A$. It is also possible to define a *stochastic* policy as $\pi : S \times A \rightarrow [0,1]$ such that for each state $s \in S$, it holds that $\pi(s,a) \geq 0$ and $\sum_{a \in A} \pi(s,a) = 1$. We will assume deterministic policies in this book unless stated otherwise.

Application of a policy to an MDP is done in the following way. First, a start state s_0 from the initial state distribution I is generated. Then, the policy π suggest the action $a_0 = \pi(s_0)$ and this action is performed. Based on the transition function T and reward function R , a transition is made to state s_1 , with probability $T(s_0, a_0, s_1)$ and a reward $r_0 = R(s_0, a_0, s_1)$ is received. This process continues, producing $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, \dots$. If the task is episodic, the process ends in state s_{goal} and is restarted in a new state drawn from I . If the task is continuing, the sequence of states can be extended indefinitely.

The policy is part of the agent and its aim is to *control* the environment modelled as an MDP. A fixed policy induces a stationary transition distribution over the MDP which can be transformed into a *Markov system*³ $\langle S', T' \rangle$ where $S' = S$ and $T'(s, s') = T(s, a, s')$ whenever $\pi(s) = a$.

1.3.3 Optimality Criteria and Discounting

In the previous sections, we have defined the environment (the MDP) and the agent (i.e. the controlling element, or policy). Before we can talk about algorithms for computing *optimal* policies, we have to define what that means. That is, we have to define what the *model of optimality* is. There are two ways of looking at optimality. First, there is the aspect of *what* is actually being optimized, i.e. what is the *goal* of the agent? Second, there is the aspect of how optimal the way in which the goal is being optimized, is. The first aspect is related to *gathering reward* and is

³ In other words, if π is fixed, the system behaves as a stochastic transition system with a stationary distribution over states.

treated in this section. The second aspect is related to the efficiency and optimality of algorithms, and this is briefly touched upon and dealt with more extensively in Section 1.5 and further.

The goal of learning in an MDP is to gather rewards. If the agent was only concerned about the immediate reward, a simple optimality criterion would be to optimize $E[r_t]$. However, there are several ways of taking into account the future in how to behave now. There are basically three models of optimality in the MDP, which are sufficient to cover most of the approaches in the literature. They are strongly related to the types of tasks that were defined in Section 1.3.1.

$$\begin{array}{ccc} E\left[\sum_{t=0}^h r_t\right] & E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] & \lim_{h \rightarrow \infty} E\left[\frac{1}{h} \sum_{t=0}^h r_t\right] \end{array}$$

Fig. 1.2 Optimality: a) finite horizon, b) discounted, infinite horizon, c) average reward

The *finite horizon* model simply takes a finite horizon of length h and states that the agent should optimize its expected reward over this horizon, i.e. the next h steps (see Figure 1.2a)). One can think of this in two ways. The agent could in the first step take the *h-step optimal action*, after this the *(h - 1)-step optimal action*, and so on. Another way is that the agent will always take the *h-step optimal action*, which is called *receding-horizon control*. The problem, however, with this model, is that the (optimal) choice for the horizon length h is not always known.

In the *infinite-horizon model*, the long-run reward is taken into account, but the rewards that are received in the future are discounted according to how far away in time they will be received. A *discount factor* γ , with $0 \leq \gamma < 1$ is used for this (see Figure 1.2b)). Note that in this *discounted* case, rewards obtained later are discounted more than rewards obtained earlier. Additionally, the discount factor ensures that – even with infinite horizon – the sum of the rewards obtained is finite. In episodic tasks, i.e. in tasks where the horizon is finite, the discount factor is not needed or can equivalently be set to 1. If $\gamma = 0$ the agent is said to be *myopic*, which means that it is only concerned about immediate rewards. The discount factor can be interpreted in several ways; as an interest rate, probability of living another step, or the mathematical trick for bounding the infinite sum. The discounted, infinite-horizon model is mathematically more convenient, but conceptually similar to the finite horizon model. Most algorithms in this book use this model of optimality.

A third optimality model is the *average-reward* model, maximizing the long-run *average reward* (see Figure 1.2c)). Sometimes this is called the *gain optimal* policy and in the limit, as the discount factor approaches 1, it is equal to the infinite-horizon discounted model. A difficult problem with this criterion that for long (or, infinite) episodes we cannot distinguish between two policies in which one receives a lot of reward in the initial phases and another one which does not. This initial difference in reward is hidden in the long-run average. This problem can be solved in using a *bias*

optimal model in which the long-run average is still being optimized, but policies are preferred if they additionally get initially extra reward. See Mahadevan (1996) for a survey on average reward RL.

Choosing between these optimality criteria can be related to the learning problem. If the length of the episode is known, the finite-horizon model is best. However, often this is not known, or the task is continuing, and then the infinite-horizon model is more suitable. Koenig and Liu (2002) gives an extensive overview of different modelings of MDPs and their relationship with optimality.

The second kind of optimality in this section is related to the more general aspect of the optimality of the learning process itself. We will encounter various concepts in the remainder of this book. We will briefly summarize three important notions here.

Learning optimality can be explained in terms of *what* the end result of learning might be. A first concern is whether the agent is able to obtain *optimal performance* in principle. For some algorithms there are proofs stating this, but for some not. In other words, is there a way to ensure that the learning process will reach a global optimum, or merely a local optimum, or even an oscillation between performances? A second kind of optimality is related to the *speed* of converging to a solution. We can distinguish between two learning methods by looking at how many interactions are needed, or how much computation is needed per interaction. And related to that, what will the performance be after a certain period of time? In supervised learning the optimality criterion is often defined in terms of *predictive accuracy* which is different from optimality in the MDP setting. Also, it is important to look at how much *experimentation* is necessary, or even allowed, for reaching optimal behavior. For example, a learning robot or helicopter might not be allowed to make many mistakes during learning. A last kind of optimality is related to how much reward is *not* obtained by the learned policy, as compared to an optimal one. This is usually called the *regret* of a learning system.

1.4 Value Functions and Bellman Equations

In the preceding sections we have defined MDPs and optimality criteria that can be useful for learning optimal policies. In this section we define *value functions*, which are a way to link the optimality criteria to policies. Most learning algorithms for MDPs compute optimal policies by learning value functions. A value function represents an estimate *how good* it is for the agent to be in a certain state (or how good it is to perform a certain action in that state). The notion of *how good* is expressed in terms of an optimality criterion, i.e. in terms of the expected return. Value functions are defined for particular policies.

The *value of a state s under policy π* , denoted $V^\pi(s)$ is the expected return when starting in s and following π thereafter. We will use the infinite-horizon, discounted

model in this section, such that this can be expressed⁴ as:

$$V^\pi(s) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \right\} \quad (1.1)$$

A similar *state-action value function* $Q : S \times A \rightarrow \mathbb{R}$ can be defined as the expected return starting from state s , taking action a and thereafter following policy π :

$$Q^\pi(s, a) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a \right\}$$

One fundamental property of value functions is that they satisfy certain recursive properties. For any policy π and any state s the expression in Equation 1.1 can recursively be defined in terms of a so-called *Bellman Equation* Bellman (1957):

$$\begin{aligned} V^\pi(s) &= E_\pi \left\{ r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots | s_t = t \right\} \\ &= E_\pi \left\{ r_t + \gamma V^\pi(s_{t+1}) | s_t = s \right\} \\ &= \sum_{s'} T(s, \pi(s), s') \left(R(s, a, s') + \gamma V^\pi(s') \right) \end{aligned} \quad (1.2)$$

It denotes that the expected value of state is defined in terms of the immediate reward and values of possible next states weighted by their transition probabilities, and additionally a discount factor. V^π is the unique solution for this set of equations. Note that multiple policies can have the same value function, but for a given policy π , V^π is unique.

The goal for any given MDP is to find a *best* policy, i.e. the policy that receives the most reward. This means maximizing the value function of Equation 1.1 for all states $s \in S$. An *optimal policy*, denoted π^* , is such that $V^{\pi^*}(s) \geq V^\pi(s)$ for all $s \in S$ and all policies π . It can be proven that the optimal solution $V^* = V^{\pi^*}$ satisfies the following Equation:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma V^*(s') \right) \quad (1.3)$$

This expression is called the *Bellman optimality equation*. It states that the value of a state under an optimal policy must be equal to the expected return for the best action in that state. To select an optimal action given the optimal state value function V^* the following rule can be applied:

$$\pi^*(s) = \arg \max_a \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma V^*(s') \right) \quad (1.4)$$

⁴ Note that we use E_π for the *expected value under policy π* .

We call this policy the *greedy policy*, denoted $\pi_{\text{greedy}}(V)$ because it greedily selects the best action using the value function V . An analogous optimal state-action value is:

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right)$$

Q -functions are useful because they make the weighted summation over different alternatives (such as in Equation 1.4) using the transition function unnecessary. No forward-reasoning step is needed to compute an optimal action in a state. This is the reason that in model-free approaches, i.e. in case T and R are unknown, Q -functions are learned instead of V -functions. The relation between Q^* and V^* is given by

$$Q^*(s,a) = \sum_{s' \in S} T(s,a,s') \left(R(s,a,s') + \gamma V^*(s') \right) \quad (1.5)$$

$$V^*(s) = \max_a Q^*(s,a) \quad (1.6)$$

Now, analogously to Equation 1.4, optimal action selection can be simply put as:

$$\pi^*(s) = \arg \max_a Q^*(s,a) \quad (1.7)$$

That is, the best action is the action that has the highest expected utility based on possible next states resulting from taking that action. One can, analogous to the expression in Equation 1.4, define a greedy policy $\pi_{\text{greedy}}(Q)$ based on Q . In contrast to $\pi_{\text{greedy}}(V)$ there is no need to consult the model of the MDP; the Q -function suffices.

1.5 Solving Markov Decision Processes

Now that we have defined MDPs, policies, optimality criteria and value functions, it is time to consider the question of *how* to compute optimal policies. *Solving* a given MDP means computing an optimal policy π^* . Several dimensions exist along which algorithms have been developed for this purpose. The most important distinction is that between *model-based* and *model-free* algorithms.

Model-based algorithms exist under the general name of DP. The basic assumption in these algorithms is that a *model* of the MDP is known beforehand, and can be used to compute value functions and policies using the Bellman equation (see Equation 1.3). Most methods are aimed at computing state value functions which can, in the presence of the model, be used for optimal action selection. In this chapter we will focus on *iterative* procedures for computing value functions and policies.

Model-free algorithms, under the general name of RL, do not rely on the availability of a perfect model. Instead, they rely on *interaction* with the environment, i.e. a *simulation* of the policy thereby generating *samples* of state transitions and rewards. These samples are then used to estimate state-action value functions.

Because a model of the MDP is not known, the agent has to *explore* the MDP to obtain information. This naturally induces a *exploration-exploitation* trade-off which has to be balanced to obtain an optimal policy. In model-based reinforcement learning, the agent does not possess an a priori a model of the environment, but estimates it while it is learning. After inducing a reasonable model of the environment, the agent can then apply dynamic programming-like algorithms to compute a policy.

A very important underlying mechanism, the so-called *generalized policy iteration* (GPI) principle, present in all methods is depicted in Figure 1.3. This principle consists of two interaction processes. The *policy evaluation* step estimates the utility of the current policy π , that is, it computes V^π . There are several ways for computing this. In model-based algorithms, one can use the model to compute it directly or iteratively approximate it. In model-free algorithms, one can *simulate* the policy and estimate its utility from the sampled execution traces. The main purpose of this step is to gather information about the policy for computing the second step, the *policy improvement* step. In this step, the values of the actions are evaluated for every state, in order to find possible improvements, i.e. possible other actions in particular states that are better than the action the current policy proposes. This step computes an improved policy π' from the current policy π using the information in V^π . Both the evaluation and the improvement steps can be implemented in various ways, and interleaved in several distinct ways. The bottom line is that there is a policy that drives value learning, i.e. it determines the value function, but in turn there is a value function that can be used by the policy to select good actions. Note that it is also possible to have an *implicit* representation of the policy, which means that only the value function is stored, and a policy is computed on-the-fly for each state based on the value function when needed. This is common practice in model-free algorithms (see Section 1.7). And vice versa it is also possible to have implicit representations of value functions in the context of an explicit policy representation. Another interesting aspect is that in general a value function does not have to be perfectly accurate. In many cases it suffices that sufficient distinction is present between suboptimal and optimal actions, such that small errors in values do not have to influence policy optimality. This is also important in *approximation* and *abstraction* methods.

Planning as an RL Problem

The MDP formalism is a general formalism for *decision-theoretic planning*, which entails that standard (deterministic) *planning* problems can be formalized as such too. All the algorithms in this chapter can – in principle – be used for these planning problems too. In order to solve planning problems in the MDP framework we have to specify *goals* and *rewards*. We can assume that the transition function T is given, accompanied by a *precondition function*. In planning we are given a *goal* function $G : S \rightarrow \{\text{true}, \text{false}\}$ that defines which states are goal states. The planning task is compute a sequence of actions $a_t, a_{t+1}, \dots, a_{t+n}$ such that applying this sequence from a *start* state will lead to a state $s \in G$. All transitions are assumed to

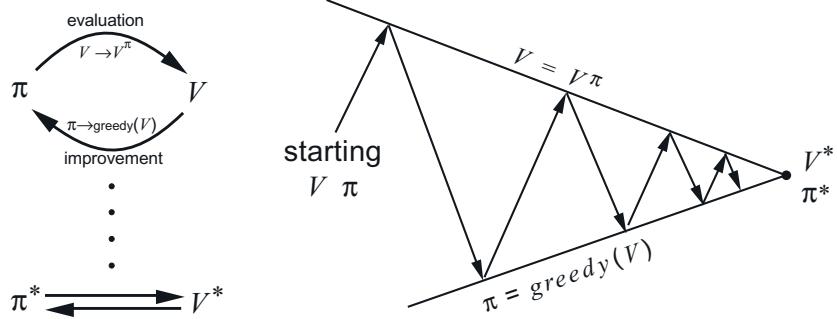


Fig. 1.3 a) The algorithms in Section 1.5 can be seen as instantiations of *Generalized Policy Iteration* (GPI) Sutton and Barto (1998). The *policy evaluation* step estimates V^π , the policy's performance. The *policy improvement* step improves the policy π based on the estimates in V^π . b) The gradual convergence of both the value function and the policy to optimal versions.

be deterministic, i.e. for all states $s \in S$ and actions $a \in A$ there exists only one state $s' \in S$ such that $T(s,a,s') = 1$. All states in G are assumed to be absorbing. The only thing left is to specify the reward function. We can specify this in such a way that a positive reinforcement is received once a goal state is reached, and zero otherwise:

$$R(s_t, a_t, s_{t+1}) = \begin{cases} 1, & \text{if } s_t \notin G \text{ and } s_{t+1} \in G \\ 0, & \text{otherwise} \end{cases}$$

Now, depending on whether the transition function and reward function are known to the agent, one can solve this planning task with either model-based or model-free learning. The difference with classic planning is that the learned policy will apply to all states.

1.6 Dynamic Programming: Model-Based Solution Techniques

The term DP refers to a class of algorithms that is able to compute optimal policies in the presence of a perfect model of the environment. The assumption that a model is available will be hard to ensure for many applications. However, we will see that from a theoretical viewpoint, as well as from an algorithmic viewpoint, DP algorithms are very relevant because they define fundamental computational mechanisms which are also used when no model is available. The methods in this section all assume a standard MDP $\langle S, A, T, R \rangle$, where the state and action sets are finite and discrete such that they can be stored in tables. Furthermore, transition, reward and value functions are assumed to store values for all states and actions separately.

1.6.1 Fundamental DP Algorithms

Two core DP methods are *policy iteration* Howard (1960) and *value iteration* Bellman (1957). In the first, the GPI mechanism is clearly separated into two steps, whereas the second represents a tight integration of policy evaluation and improvement. We will consider both these algorithms in turn.

1.6.1.1 Policy Iteration

Policy iteration (PI) Howard (1960) iterates between the two phases of GPI. The *policy evaluation* phase computes the value function of the current policy and the *policy improvement* phase computes an improved policy by a maximization over the value function. This is repeated until converging to an optimal policy.

Policy Evaluation: The Prediction Problem.

A first step is to find the value function V^π of a fixed policy π . This is called the *prediction problem*. It is a part of the complete problem, that of computing an *optimal* policy. Remember from the previous sections that for all $s \in S$,

$$V^\pi(s) = \sum_{s' \in S} T(s, \pi(s), s') \left(R(s, \pi(s), s') + \gamma V^\pi(s') \right) \quad (1.8)$$

If the dynamics of the system are known, i.e. a model of the MDP is given, then these equations form a system of $|S|$ equations in $|S|$ unknowns (the values of V^π for each $s \in S$). This can be solved by linear programming (LP). However, an *iterative* procedure is possible, and in fact common in DP and RL. The Bellman equation is transformed into an *update rule* which updates the current value function V_k^π into V_{k+1}^π by 'looking one step further in the future', thereby extending the planning horizon with one step:

$$\begin{aligned} V_{k+1}^\pi(s) &= E_\pi \left\{ r_t + \gamma V_k^\pi(s_{t+1}) | s_t = s \right\} \\ &= \sum_{s'} T(s, \pi(s), s') \left(R(s, \pi(s), s') + \gamma V_k^\pi(s') \right) \end{aligned} \quad (1.9)$$

The sequence of approximations of V_k^π as k goes to infinity can be shown to converge. In order to converge, the update rule is applied to each state $s \in S$ in each iteration. It replaces the old value for that state by a new one that is based on the expected value of possible successor states, intermediate rewards and weighted by the transition probabilities. This operation is called a *full backup* because it is based on all possible transitions from that state.

A more general formulation can be given by defining a *backup operator* B^π over arbitrary real-valued functions φ over the state space (e.g. a value function):

$$(B^\pi \varphi)(s) = \sum_{s' \in S} T(s, \pi(s), s') \left(R(s, \pi(s), s') + \gamma \varphi(s') \right) \quad (1.10)$$

The value function V^π of a fixed policy π satisfies the *fixed point* of this backup operator as $V^\pi = B^\pi V^\pi$. A useful special case of this backup operator is defined with respect to a fixed action a :

$$(B^a \varphi)(s) = R(s) + \gamma \sum_{s' \in S} T(s, a, s') \varphi(s')$$

Now LP for solving the *prediction problem* can be stated as follows. Computing V^π can be accomplished by solving the Bellman equations (see Equation 1.3) for all states. The *optimal* value function V^* can be found by using a LP problem solver that computes $V^* = \arg \max_V \sum_s V(s)$ subject to $V(s) \geq (B^a V)(s)$ for all a and s .

Policy Improvement

Now that we know the value function V^π of a policy π as the outcome of the policy evaluation step, we can try to improve the policy. First we identify the value of all actions by using:

$$Q^\pi(s, a) = E_\pi \left\{ r_t + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a \right\} \quad (1.11)$$

$$= \sum_{s'} T(s, a, s') \left(R(s, a, s') + \gamma V^\pi(s') \right) \quad (1.12)$$

If now $Q^\pi(s, a)$ is larger than $V^\pi(s)$ for some $a \in A$ then we could do better by choosing action a instead of the current $\pi(s)$. In other words, we can *improve* the current policy by selecting a different, better, action in a particular state. In fact, we can evaluate all actions in all states and choose the best action in all states. That is, we can compute the *greedy* policy π' by selecting the best action in each state, based on the current value function V^π :

$$\begin{aligned} \pi'(s) &= \arg \max_a Q^\pi(s, a) \\ &= \arg \max_a E \left\{ r_t + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a \right\} \\ &= \arg \max_a \sum_{s'} T(s, a, s') \left(R(s, a, s') + \gamma V^\pi(s') \right) \end{aligned} \quad (1.13)$$

Computing an improved policy by greedily selecting the best action with respect to the value function of the original policy is called *policy improvement*. If the policy

```

Require:  $V(s) \in \mathbb{R}$  and  $\pi(s) \in A(s)$  arbitrarily for all  $s \in S$ 
{POLICY EVALUATION}
repeat
     $\Delta := 0$ 
    for each  $s \in S$  do
         $v := V^\pi(s)$ 
         $V(s) := \sum_{s'} T(s, \pi(s), s') \left( R(s, \pi(s), s') + \gamma V(s') \right)$ 
         $\Delta := \max(\Delta, |v - V(s)|)$ 
    until  $\Delta < \sigma$ 
{POLICY IMPROVEMENT}
policy-stable := true
for each  $s \in S$  do
     $b := \pi(s)$ 
     $\pi(s) := \arg \max_a \sum_{s'} T(s, a, s') \left( R(s, a, s') + \gamma \cdot V(s') \right)$ 
    if  $b \neq \pi(s)$  then policy-stable := false
if policy-stable then stop; else go to POLICY EVALUATION

```

Algorithm 1. Policy Iteration Howard (1960)

cannot be improved in this way, it means that the policy is already optimal and its value function satisfies the Bellman equation for the optimal value function. In a similar way one can also perform these steps for stochastic policies by blending the action probabilities into the expectation operator.

Summarizing, *policy iteration* Howard (1960) starts with an arbitrary initialized policy π_0 . Then a sequence of iterations follows in which the current policy is evaluated after which it is improved. The first step, the *policy evaluation* step computes V^{π_k} , making use of Equation 1.9 in an iterative way. The second step, the *policy improvement* step, computes π_{k+1} from π_k using V^{π_k} . For each state, using equation 1.4, the optimal action is determined. If for all states s , $\pi_{k+1}(s) = \pi_k(s)$, the policy is *stable* and the policy iteration algorithm can stop. Policy iteration generates a sequence of alternating policies and value functions

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \pi_2 \rightarrow V^{\pi_2} \rightarrow \pi_3 \rightarrow V^{\pi_3} \rightarrow \dots \rightarrow \pi^*$$

The complete algorithm can be found in Algorithm 1.

For finite MDPs, i.e. state and action spaces are finite, policy iteration converges after a finite number of iterations. Each policy π_{k+1} is a strictly better policy than π_k unless $\pi_k = \pi^*$, in which case the algorithm stops. And because for a finite MDP, the number of different policies is finite, policy iteration converges in finite time. In practice, it usually converges after a small number of iterations. Although policy iteration computes the optimal policy for a given MDP in finite time, it is relatively inefficient. In particular the first step, the policy evaluation step, is computationally expensive. Value functions for all intermediate policies $\pi_0, \dots, \pi_k, \dots, \pi^*$ are computed, which involves multiple sweeps through the complete state space per iteration. A bound on the number of iterations is not known Littman et al (1995) and

```

Require: initialize  $V$  arbitrarily (e.g.  $V(s) := 0, \forall s \in S$ )
repeat
     $\Delta := 0$ 
    for each  $s \in S$  do
         $v := V(s)$ 
        for each  $a \in A(s)$  do
            
$$Q(s,a) := \sum_{s'} T(s,a,s') \left( R(s,a,s') + \gamma V(s') \right)$$

             $V(s) := \max_a Q(s,a)$ 
             $\Delta := \max(\Delta, |v - V(s)|)$ 
    until  $\Delta < \sigma$ 

```

Algorithm 2. Value Iteration Bellman (1957)

depends on the MDP transition structure, but it often converges after few iterations in practice.

1.6.1.2 Value Iteration

The policy iteration algorithm completely separates the evaluation and improvement phases. In the evaluation step, the value function must be computed in the limit. However, it is not necessary to wait for full convergence, but it is possible to stop evaluating earlier and improve the policy based on the evaluation so far. The extreme point of truncating the evaluation step is the *value iteration* Bellman (1957) algorithm. It breaks off evaluation after just one iteration. In fact, it immediately blends the policy improvement step into its iterations, thereby purely focusing on estimating directly the value function. Necessary updates are computed on-the-fly. In essence, it combines a truncated version of the policy evaluation step with the policy improvement step, which is essentially Equation 1.3 turned into one update rule:

$$V_{t+1}(s) = \max_a \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma V_t(s') \right) \quad (1.14)$$

$$= \max_a Q_{t+1}(s,a). \quad (1.15)$$

Using Equations (1.14) and (1.15), the value iteration algorithm (see Figure 2) can be stated as follows: starting with a value function V_0 over all states, one iteratively updates the value of each state according to (1.14) to get the next value functions V_t ($t = 1, 2, 3, \dots$). It produces the following sequence of value functions:

$$V_0 \rightarrow V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_4 \rightarrow V_5 \rightarrow V_6 \rightarrow V_7 \rightarrow \dots V^*$$

Actually, in the way it is computed it also produces the intermediate Q -value functions such that the sequence is

$$V_0 \rightarrow Q_1 \rightarrow V_1 \rightarrow Q_2 \rightarrow V_2 \rightarrow Q_3 \rightarrow V_3 \rightarrow Q_4 \rightarrow V_4 \rightarrow \dots V^*$$

Value iteration is guaranteed to converge in the limit towards V^* , i.e. the Bellman optimality Equation (1.3) holds for each state. A deterministic policy π for all states $s \in S$ can be computed using Equation 1.4. If we use the same general *backup operator* mechanism used in the previous section, we can define value iteration in the following way.

$$(B^* \varphi)(s) = \max_a \sum_{s' \in S} T(s,a,s') \left\{ R(s,a,s') + \gamma \varphi(s') \right\} \quad (1.16)$$

The backup operator B^* functions as a *contraction mapping* on the value function. If we let π^* denote the *optimal* policy and V^* its value function, we have the relationship (fixed point) $V^* = B^*V^*$ where $(B^*V)(s) = \max_a (B^aV)(s)$. If we define $Q^*(s,a) = B^aV^*$ then we have $\pi^*(s) = \pi_{\text{greedy}}(V^*)(s) = \arg \max_a Q^*(s,a)$. That is, the algorithm starts with an arbitrary value function V^0 after which it iterates $V_{t+1} = B^*V^t$ until $\|V_{t+1} - V_t\|_S < \varepsilon$, i.e. until the distance between subsequent value function approximations is small enough.

1.6.2 Efficient DP Algorithms

The policy iteration and value iteration algorithms can be seen as spanning a *spectrum* of DP approaches. This spectrum ranges from complete *separation* of evaluation and improvement steps to a complete *integration* of these steps. Clearly, in between the extreme points is much room for variations on algorithms. Let us first consider the computational complexity of the extreme points.

Complexity

Value iteration works by producing successive approximations of the optimal value function. Each iteration can be performed in $O(|A||S|^2)$ steps, or faster if T is sparse. However, the *number* of iterations can grow exponentially in the discount factor cf. Bertsekas and Tsitsiklis (1996). This follows from the fact that a larger γ implies that a longer sequence of future rewards has to be taken into account, hence a larger number of value iteration steps because each step only extends the horizon taking into account in V by one step. The complexity of value iteration is linear in number of actions, and quadratic in the number of states. But, usually the transition matrix is sparse. In practice policy iteration converges much faster, but each evaluation step is expensive. Each iteration has a complexity of $O(|A||S|^2 + |S|^3)$, which can grow large quickly. A worst case bound on the number of iterations is not known Littman et al (1995). Linear programming is a common tool that can be used for the evaluation too. In general, the number of iterations and value backups can quickly

grow extremely large when the problem size grows. The state spaces of games such as backgammon and chess consist of too many states to perform just one full sweep. In this section we will describe some efficient variations on DP approaches. Detailed coverage of complexity results for the solution of MDPs can be found in Littman et al (1995); Bertsekas and Tsitsiklis (1996); Boutilier et al (1999).

The efficiency of DP can be roughly improved along two lines. The first is a *tighter integration* of the evaluation and improvement steps of the GPI process. We will discuss this issue briefly in the next section. The second is that of using (*heuristic*) *search* algorithms in combination with DP algorithms. For example, using search as an exploration mechanism can highlight important parts of the state space such that value backups can be concentrated on these parts. This is the underlying mechanism used in the methods discussed briefly in Section 1.6.2.2

1.6.2.1 Styles of Updating

The full backups updates in DP algorithms can be done in several ways. We have assumed in the description of the algorithms that in each step an old and a new value function are kept in memory. Each update puts a new value in the new table, based on the information of the old. This is called *synchronous*, or *Jacobi-style* updating Sutton and Barto (1998). This is useful for explanation of algorithms and theoretical proofs of convergence. However, there are two more common ways for updates. One can keep a single table and do the updating directly in there. This is called *in-place* updating Sutton and Barto (1998) or *Gauss-Seidel* Bertsekas and Tsitsiklis (1996) and usually speeds up convergence, because during one sweep of updates, some updates use already newly updated values of other states. Another type of updating is called *asynchronous updating* which is an extension of the *in-place* updates, but here updates can be performed in any order. An advantage is that the updates may be distributed unevenly throughout the state(-action) space, with more updates being given to more important parts of this space. For all these methods convergence can be proved under the general condition that values are updated infinitely often but with a finite frequency.

Modified Policy Iteration

Modified policy iteration (MPI) Puterman and Shin (1978) strikes a middle ground between value and policy iteration. MPI maintains the two separate steps of GPI, but both steps are not necessarily computed in the limit. The key insight here is that for policy improvement, one does not need an *exactly* evaluated policy in order to improve it. For example, the policy estimation step can be approximative after which a policy improvement step can follow. In general, both steps can be performed quite independently *by different means*. For example, instead of iteratively applying the Bellman update rule from Equation 1.15, one can perform the policy estimation step by using a sampling procedure such as *Monte Carlo* estimation Sutton and Barto

(1998). These general forms with mixed forms of estimation and improvements is captured by the generalized policy iteration mechanism depicted in Figure 1.3. Policy iteration and value iteration are both the extreme cases of modified policy iteration, whereas MPI is a general method for asynchronous updating.

1.6.2.2 Heuristics and Search

In many realistic problems, only a fraction of the state space is relevant to the problem of reaching the goal state from some state s . This has inspired a number of algorithms that focus computation on states that seem most relevant for finding an optimal policy from a start state s . These algorithms usually display good *anytime behavior*, i.e. they produce good or reasonable policies fast, after which they are gradually improved. In addition, they can be seen as implementing various ways of *asynchronous DP*.

Envelopes and Fringe States

One form of *asynchronous* methods is the PLEXUS system Dean et al (1995). It was designed for goal-based reward functions, i.e. episodic tasks in which only goal states get positive reward. It starts with an approximated version of the MDP in which not the full state space is contained. This smaller version of the MDP is called an *envelope* and it includes the agent's current state and the goal state. A special OUT state represents all the states outside the envelope. The initial envelope is constructed by a forward search until a goal state is found. The envelope can be extended by considering states outside the envelope that can be reached with high probability. The intuitive idea is to include in the envelope all the states that are likely to be reached on the way to the goal. Once the envelope has been constructed a policy is computed through policy iteration. If at any point the agent leaves the envelope, it has to replan by extending the envelope. This combination of learning and planning still uses policy iteration, but on a much smaller (and presumably more relevant with respect to the goal) state space.

A related method proposed in Tash and Russell (1994) considers goal-based tasks too. However, instead of the single OUT state, they keep a *fringe* of states on the edge of the envelope and use a heuristic to estimate values of the other states. When computing a policy for the envelope, all fringe states become absorbing states with the heuristic set as their value. Over time the heuristic values of the fringe states converge to the optimal values of those states.

Similar to the previous methods the LAO* algorithm Hansen and Zilberstein (2001) also alternates between an expansion phase and a policy generation phase. It too keeps a fringe of states outside the envelope such that expansions can be larger than the envelope method in Dean et al (1995). The motivation behind LAO* was to extend the classic search algorithm AO* cf. Russell and Norvig (2003) to *cyclic* domains such as MDPs.

Search and Planning in DP

Real-time dynamic DP (RTDP) Barto et al (1995) combines forward search with DP too. It is used as an alternative for value iteration in which only a subset of values in the state space are backed up in each iteration. RTDP performs *trials* from a randomly selected state to a goal state, by simulating the greedy policy using an *admissible heuristic* function as the initial value function. It then backups values *fully* only along these trials, such that backups are concentrated on the *relevant* parts of the state space. The approach was later extended into *labeled* RTDP Bonet and Geffner (2003b) where some states are *labeled* as *solved* which means that their value has already converged. Furthermore, it was recently extended to *bounded* RTDPMcMahan et al (2005) which keeps lower and upper *bounds* on the optimal value function. Other recent methods along these lines are *focussed* DP Ferguson and Stentz (2004) and *heuristic search-DP* Bonet and Geffner (2003a).

1.7 Reinforcement Learning: Model-Free Solution Techniques

The previous section has reviewed several methods for computing an optimal policy for an MDP assuming that a (perfect) model is available. RL is primarily concerned with how to obtain an optimal policy when such a model is not available. RL adds to MDPs a focus on approximation and incomplete information, and the need for sampling and exploration. In contrast with the algorithms discussed in the previous section, *model-free* methods do not rely on the availability of priori known transition and reward models, i.e. a *model of the MDP*. The lack of a model generates a need to *sample* the MDP to gather statistical knowledge about this unknown model. Many model-free RL techniques exist that probe the environment by doing actions, thereby estimating the same kind of state value and state-action value functions as model-based techniques. This section will review model-free methods along with several efficient extensions.

In model-free contexts one has still a choice between two options. The first one is first to *learn* the transition and reward model from interaction with the environment. After that, when the model is (approximately or sufficiently) correct, all the DP methods from the previous section apply. This type of learning is called *indirect* or model-based RL. The second option, called *direct* RL, is to step right into estimating values for actions, without even estimating the model of the MDP. Additionally, mixed forms between these two exist too. For example, one can still do model-free estimation of action values, but use an approximated model to speed up value learning by using this model to perform more, and in addition, full backups of values (see Section 1.7.3). Most model-free methods however, focus on direct estimation of (action) values.

A second choice one has to make is what to do with the *temporal credit assignment problem*. It is difficult to assess the utility of some action, if the real effects of this particular action can only be perceived much later. One possibility is to wait

```

for each episode do
     $s \in S$  is initialized as the starting state
     $t := 0$ 
    repeat
        choose an action  $a \in A(s)$ 
        perform action  $a$ 
        observe the new state  $s'$  and received reward  $r$ 
        update  $\tilde{T}$ ,  $\tilde{R}$ ,  $\tilde{Q}$  and/or  $\tilde{V}$ 
        using the experience  $\langle s, a, r, s' \rangle$ 
         $s := s'$ 
    until  $s'$  is a goal state

```

Algorithm 3. A general algorithm for online RL

until the "end" (e.g. of an episode) and punish or reward specific actions along the path taken. However, this will take a lot of memory and often, with ongoing tasks, it is not known beforehand whether, or when, there will be an "end". Instead, one can use similar mechanisms as in *value iteration* to adjust the estimated value of a state based on the immediate reward and the estimated (discounted) value of the next state. This is generally called *temporal difference learning* which is a general mechanism underlying the model-free methods in this section. The main difference with the update rules for DP approaches (such as Equation 1.14) is that the transition function T and reward function R cannot appear in the update rules now. The general class of algorithms that interact with the environment and update their estimates after each experience is called *online RL*.

A general template for *online RL* is depicted in Figure 3. It shows an interaction loop in which the agent selects an action (by whatever means) based on its current state, gets feedback in the form of the resulting state and an associated reward, after which it updates its estimated values stored in \tilde{V} and \tilde{Q} and possibly statistics concerning \tilde{T} and \tilde{R} (in case of some form of indirect learning). The selection of the action is based on the current state s and the value function (either Q or V). To solve the exploration-exploitation problem, usually a separate *exploration* mechanism ensures that sometimes the best action (according to current estimates of action values) is taken (exploitation) but sometimes a different action is chosen (exploration). Various choices for exploration, ranging from random to sophisticated, exist and we will see some examples below and in Section 1.7.3.

Exploration

One important aspect of model-free algorithms is that there is a need for *exploration*. Because the model is unknown, the learner has to try out different actions to see their results. A learning algorithm has to strike a balance between *exploration* and *exploitation*, i.e. in order to gain a lot of reward the learner has to exploit its current knowledge about good actions, although it sometimes must try out different actions to explore the environment for finding possible better actions. The most

basic exploration strategy is the ϵ -greedy policy, i.e. the learner takes its current best action with probability $(1 - \epsilon)$ and a (randomly selected) other action with probability ϵ . There are many more ways of doing exploration (see Wiering (1999); Reynolds (2002); Ratitch (2005) for overviews) and in Section 1.7.3 we will see some examples. One additional method that is often used in combination with the algorithms in this section is the *Boltzmann* (or: *softmax*) exploration strategy. It is only slightly more complicated than the ϵ -greedy strategy. The action selection strategy is still random, but selection probabilities are weighted by their relative Q -values. This makes it more likely for the agent to choose very good actions, whereas two actions that have similar Q -values will have almost the same probability to get selected. Its general form is

$$P(a) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_i e^{\frac{Q(s,a_i)}{T}}} \quad (1.17)$$

in which $P(a)$ is the probability of selecting action a and T is the *temperature* parameter. Higher values of T will move the selection strategy more towards a purely random strategy and lower values will move to a fully greedy strategy. A combination of both ϵ -greedy and Boltzmann exploration can be taken by taking the best action with probability $(1 - \epsilon)$ and otherwise an action computed according to Equation 1.17 Wiering (1999).

Another simple method to stimulate exploration is *optimistic Q-values initialization*; one can initialize all Q -values to high values – e.g. an a priori defined upper-bound – at the start of learning. Because Q -values will decrease during learning, actions that have not been tried a number of times will have a large enough value to get selected when using Boltzmann exploration for example. Another solution with a similar effect is to keep counters on the number of times a particular state-action pair has been selected.

1.7.1 Temporal Difference Learning

Temporal difference learning algorithms learn estimates of values based on other estimates. Each step in the world generates a *learning example* which can be used to bring some value in accordance to the immediate reward and the estimated value of the next state or state-action pair. An intuitive example, along the lines of Sutton and Barto (1998)(Chapter 6), is the following.

Imagine you have to predict at what time your guests can arrive for a small diner in your house. Before cooking, you have to go to the supermarket, the butcher and the wine seller, in that order. You have estimates of driving times between all locations, and you predict that you can manage to visit the two last stores both in 10 minutes, but given the crowded time on the day, your estimate about the supermarket is a half hour. Based on this prediction, you have notified your guests that they can arrive no earlier than 18.00h. Once you have found out while in the supermarket that it will take you only 10 minutes to get all the things you need, you can adjust your

estimate on arriving back home with 20 minutes less. However, once on your way from the butcher to the wine seller, you see that there is quite some traffic along the way and it takes you 30 minutes longer to get there. Finally you arrive 10 minutes later than you predicted in the first place. The bottom line of this example is that you can adjust your estimate about what time you will be back home every time you have obtained new information about in-between steps. Each time you can adjust your estimate on how long it will still take based on actually experienced times of parts of your path. This is the main principle of TD learning: you do not have to wait until the end of a trial to make updates along your path.

TD methods learn their value estimates based on estimates of other values, which is called *bootstrapping*. They have an advantage over DP in that they do not require a model of the MDP. Another advantage is that they are naturally implemented in an online, incremental fashion such that they can be easily used in various circumstances. No full sweeps through the full state space are needed; only along experienced paths values get updated, and updates are effected after each step.

TD(0)

TD(0) is a member of the family of TD learning algorithms Sutton (1988). It solves the prediction problem, i.e. it estimates V^π for some policy π , in an online, incremental fashion. $TD(0)$ can be used to evaluate a policy and works through the use of the following update rule⁵:

$$V_{k+1}(s) := V_k(s) + \alpha \left(r + \gamma V_k(s') - V_k(s) \right)$$

where $\alpha \in [0,1]$ is the *learning rate* that determines by how much values get updated. This backup is performed after experiencing the transition from state s to s' based on the action a , while receiving reward r . The difference with DP backups such as used in Equation 1.14 is that the update is still done by using bootstrapping, but it is based on an *observed* transition, i.e. it uses a *sample backup* instead of a full backup. Only the value of one successor state is used, instead of a weighted average of all possible successor states. When using the value function V^π for action selection, a model is needed to compute an expected value over all action outcomes (e.g. see Equation 1.4).

The learning rate α has to be decreased appropriately for learning to converge. Sometimes the learning rate can be defined for states separately as in $\alpha(s)$, in which case it can be dependent on how often the state is visited. The next two algorithms learn Q -functions directly from samples, removing the need for a transition model for action selection.

⁵ The learning parameter α should comply with some criteria on its value, and the way it is changed. In the algorithms in this section, one often chooses a small, fixed learning parameter, or it is decreased every iteration.

```

Require: discount factor  $\gamma$ , learning parameter  $\alpha$ 
initialize  $Q$  arbitrarily (e.g.  $Q(s,a) = 0, \forall s \in S, \forall a \in A$ )
for each episode do
     $s$  is initialized as the starting state
    repeat
        choose an action  $a \in A(s)$  based on  $Q$  and an exploration strategy
        perform action  $a$ 
        observe the new state  $s'$  and received reward  $r$ 
        
$$Q(s,a) := Q(s,a) + \alpha \left( r + \gamma \cdot \max_{a' \in A(s')} Q(s',a') - Q(s,a) \right)$$

         $s := s'$ 
    until  $s'$  is a goal state

```

Algorithm 4. Q -Learning Watkins and Dayan (1992)

Q -learning

One of the most basic and popular methods to estimate Q -value functions in a model-free fashion is the Q -learning algorithm Watkins (1989); Watkins and Dayan (1992), see Algorithm 4.

The basic idea in Q -learning is to incrementally estimate Q -values for actions, based on feedback (i.e. rewards) and the agent's Q -value function. The update rule is a variation on the theme of TD learning, using Q -values and a built-in max-operator over the Q -values of the next state in order to update Q_t into Q_{t+1} :

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q_k(s_{t+1}, a) - Q_k(s_t, a_t) \right) \quad (1.18)$$

The agent makes a step in the environment from state s_t to s_{t+1} using action a_t while receiving reward r_t . The update takes place on the Q -value of action a_t in the state s_t from which this action was executed.

Q -learning is exploration-insensitive. It means that it will converge to the optimal policy regardless of the exploration policy being followed, under the assumption that each state-action pair is visited an infinite number of times, and the learning parameter α is decreased appropriately Watkins and Dayan (1992); Bertsekas and Tsitsiklis (1996).

SARSA

Q -learning is an *off-policy* learning algorithm, which means that while following some exploration policy π , it aims at estimating the optimal policy π^* . A related *on-policy* algorithm that learns the Q -value function for the policy the agent is actually executing is the SARSA Rummery and Niranjan (1994); Rummery (1995); Sutton (1996) algorithm, which stands for **S**tate–**A**ction–**RS**tate–**A**ction. It uses the following update rule:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \right) \quad (1.19)$$

where the action a_{t+1} is the action that is executed by the current policy for state s_{t+1} . Note that the max-operator in Q -learning is replaced by the estimate of the value of the next action according to the policy. This learning algorithm will still converge in the limit to the optimal value function (and policy) under the condition that all states and actions are tried infinitely often and the policy converges in the limit to the greedy policy, i.e. such that exploration does not occur anymore Singh et al (2000). SARSA is especially useful in non-stationary environments. In these situations one will never reach an optimal policy. It is also useful if *function approximation* is used, because off-policy methods can diverge when this is used.

Actor-Critic Learning

Another class of algorithms that precede Q -learning and SARSA are *actor-critic* methods Witten (1977); Barto et al (1983); Konda and Tsitsiklis (2003), which learn on-policy. This branch of TD methods keeps a *separate* policy independent of the value function. The policy is called the *actor* and the value function the *critic*. The critic – typically a state-value function – evaluates, or: criticizes, the actions executed by the actor. After action selection, the critic evaluates the action using the TD-error:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

The purpose of this error is to strengthen or weaken the selection of this action in this state. A *preference* for an action a in some state s can be represented as $p(s, a)$ such that this preference can be modified using:

$$p(s_t, a_t) := p(s_t, a_t) + \beta \delta_t$$

where a parameter β determines the size of the update. There are other versions of actor-critic methods, differing mainly in how preferences are changed, or experience is used (for example using *eligibility traces*, see next section). An advantage of having a separate policy representation is that if there are many actions, or when the action space is continuous, there is no need to consider all actions' Q -values in order to select one of them. A second advantage is that they can learn *stochastic* policies naturally. Furthermore, a priori knowledge about policy constraints can be used, e.g. see Främling (2005).

Average Reward Temporal Difference Learning

We have explained Q -learning and related algorithms in the context of discounted, infinite-horizon MDPs. Q -learning can also be adapted to the average-reward framework, for example in the R -learning algorithm Schwartz (1993). Other extensions

of algorithms to the average reward framework exist (see Mahadevan (1996) for an overview).

1.7.2 Monte Carlo Methods

Other algorithms that use more *unbiased* estimates are *Monte Carlo* (MC) techniques. They keep frequency counts on state-action pairs and future reward-sums (returns) and base their values on these estimates. MC methods only require samples to estimate average sample returns. For example, in MC policy evaluation, for each state $s \in S$ all returns obtained from s are kept and the value of a state $s \in S$ is just their average. In other words, MC algorithms treat the long-term reward as a random variable and take as its estimate the sampled mean. In contrast with one-step TD methods, MC estimates values based on *averaging sample returns* observed during interaction. Especially for episodic tasks this can be very useful, because samples from complete returns can be obtained. One way of using MC is by using it for the evaluation step in policy iteration. However, because the sampling is dependent on the current policy π , only returns for actions suggested by π are evaluated. Thus, exploration is of key importance here, just as in other model-free methods.

A distinction can be made between *every-visit* MC, which averages over all *visits* of a state $s \in S$ in all episodes, and *first-visit* MC, which averages over just the returns obtained from the first visit to a state $s \in S$ for all episodes. Both variants will converge to V^π for the current policy π over time. MC methods can also be applied to the problem of estimating action values. One way of ensuring enough exploration is to use *exploring starts*, i.e. each state-action pair has a non-zero probability of being selected as the initial pair. MC methods can be used for both on-policy and off-policy control, and the general pattern complies with the generalized policy iteration procedure. The fact that MC methods do not *bootstrap* makes them less dependent on the *Markov assumption*. TD methods too focus on *sampled* experience, although they do use bootstrapping.

Learning a Model

We have described several methods for learning value functions. Indirect or model-based RL can also be used to estimate a *model* of the underlying MDP. An average over sample transition probabilities experienced during interaction can be used to gradually estimate transition probabilities. The same can be done for *immediate* rewards. *Indirect* RL algorithms make use of this to strike a balance between model-based and model-free learning. They are essentially model-free, but learn a transition and reward model in parallel with model-free RL, and use this model to do more efficient value function learning (see also the next section). An example of this is the DYNA model Sutton (1991a). Another method that often employs model learning is *prioritized sweeping* Moore and Atkeson (1993). Learning a model can

also be very useful to learn in continuous spaces where the transition model is defined over a discretized version of the underlying (infinite) state space Großmann (2000).

Relations with Dynamic Programming

The methods in this section solve essentially similar problems as DP techniques. RL approaches can be seen as *asynchronous* DP. There are some important differences in both approaches though.

RL approaches avoid the exhaustive sweeps of DP by restricting computation on, or in the neighborhood of, sampled trajectories, either real or simulated. This can exploit situations in which many states have low probabilities of occurring in actual trajectories. The backups used in DP are simplified by using sampling. Instead of generating and evaluating all of a state's possible immediate successors, the estimate of a backup's effect is done by sampling from the appropriate distribution. MC methods use this to base their estimates completely on the sample returns, without bootstrapping using values of other, sampled, states. Furthermore, the focus on learning (action) value functions in RL is easily amenable to *function approximation* approaches. Representing value functions and/or policies can be done more compactly than lookup-table representations by using *numeric regression algorithms* without breaking the standard RL interaction process; one can just feed the update values into a regression engine.

An interesting point here is the similarity between *Q*-learning and *value iteration* on the one hand and SARSA and *policy iteration* on the other hand. In the first two methods, the updates immediately combine policy evaluation and improvement into one step by using the max-operator. In contrast, the second two methods separate evaluation and improvement of the policy. In this respect, value iteration can be considered as off-policy because it aims at directly estimating V^* whereas policy iteration estimates values for the current policy and is on-policy. However, in the model-based setting the distinction is only superficial, because instead of samples that can be influenced by an on-policy distribution, a model is available such that the distribution over states and rewards is known.

1.7.3 Efficient Exploration and Value Updating

The methods in the previous section have shown that both prediction and control can be learned using samples from interaction with the environment, without having access to a model of the MDP. One problem with these methods is that they often need a large number of experiences to converge. In this section we describe a number of extensions used to speed up learning. One direction for improvement lies in the exploration. One can – in principle – use model estimation until one knows everything about the MDP but this simply takes too long. Using more information enables

more focused exploration procedures to generate experience more efficiently. Another direction is to put more efforts in using the experience for updating multiple values of the value function on each step. Improving exploration generates *better samples*, whereas improving updates will squeeze *more information* out of samples.

Efficient Exploration

We have already encountered ϵ -greedy and *Boltzmann* exploration. Although commonly used, these are relatively simple *undirected* exploration methods. They are mainly driven by *randomness*. In addition, they are *stateless*, i.e. the exploration is driven without knowing which areas of the state space have been explored so far.

A large class of *directed* methods for exploration have been proposed in the literature that use additional information about the learning process. The focus of these methods is to do more uniform exploration of the state space and to balance the relative benefits of discovering new information relative to exploiting current knowledge. Most methods use or learn a model of the MDP in parallel with RL. In addition they learn an *exploration value function*.

Several options for directed exploration are available. One distinction between methods is whether to work *locally* (e.g. exploration of individual state-action pairs) or *globally* by considering information about parts or the complete state-space when making a decision to explore. Furthermore, there are several other classes of exploration algorithms.

Counter-based or *recency-based* methods keep records of how often, or how long ago, a state-action pair has been visited. *Error-based* methods use an exploration bonus based on the error in the value of states. Other methods base exploration on the *uncertainty* about the value of a state, or the *confidence* about the state's current value. They decide whether to explore by calculating the probability that an explorative action will obtain a larger reward than expected. The *interval estimation* (IE) method Kaelbling (1993) is an example of this kind of methods. IE uses a statistical model to measure the degree of uncertainty of each $Q(s,a)$ -value. An upper bound can be calculated on the likely value of each Q -value, and the action with the highest upper bound is taken. If the action taken happens to be a poor choice, the upper bound will be decreased when the statistical model is updated. Good actions will continue to have a high upper bound and will be chosen often. In contrast to counter- and recency-based exploration, IE is concerned with *action exploration* and not with *state space* exploration. Wiering (1999) (see also Wiering and Schmidhuber (1998a)) introduced an extension to model-based RL in the *model-based interval estimation* algorithm in which the same idea is used for estimates of transition probabilities.

Another method that deals explicitly with the exploration-exploitation trade-off is the E^3 method Kearns and Singh (1998). E^3 stands for *explicit exploration and exploitation*. It learns by updating a model of the environment by collecting

statistics. The state space is divided into *known* and *unknown* parts. On every step a decision is made whether the known part contains sufficient opportunities for getting rewards or whether the unknown part should be explored to obtain possibly more reward. An important aspect of this algorithm is that it was the first general near-optimal (tabular) RL algorithm with provable bounds on computation time. The approach was extended in Brafman and Tennenholtz (2002) into the more general algorithm R-MAX. It too provides a polynomial bound on computation time for reaching near-optimal policies. As a last example, Ratitch (2005) presents an approach for efficient, directed exploration based on more sophisticated characteristics of the MDP such as an *entropy* measure over state transitions. An interesting feature of this approach is that these characteristics can be computed *before* learning and be used in combination with other exploration methods, thereby improving their behavior.

For a more detailed coverage of exploration strategies we refer the reader to Ratitch (2005) and Wiering (1999).

Guidance and Shaping

Exploration methods can be used to speed up learning and focus attention to relevant areas in the state space. The exploration methods mainly use statistics derived from the problem before or during learning. However, sometimes more information is available that can be used to *guide* the learner. For example, if a reasonable policy for a domain is available, it can be used to generate more useful learning samples than (random) exploration could do. In fact, humans are usually very bad in specifying optimal policies, but considerably good at specifying reasonable ones⁶.

The work in *behavioral cloning* Bain and Sammut (1995) takes an extreme point on the guidance spectrum in that the goal is to *replicate* example behavior from expert traces, i.e. to *clone* this *behavior*. This type of guidance moves learning more in the direction of *supervised* learning. Another way to help the agent is by *shaping* Mataric (1994); Dorigo and Colombetti (1997); Ng et al (1999). Shaping pushes the reward closer to the subgoals of behavior, and thus encourages the agent to incrementally improve its behavior by searching the policy space more effectively. This is also related to the general issue of giving rewards to appropriate subgoals, and the gradual increase in difficulty of tasks. The agent can be trained on increasingly more difficult problems, which can also be considered as a form of guidance.

Various other mechanisms can be used to provide guidance to RL algorithms, such as decompositions Dixon et al (2000), heuristic rules for better exploration Främling (2005) and various types of *transfer* in which knowledge learned in one problem is transferred to other, related problems, e.g. see Konidaris (2006).

⁶ Quote taken from the invited talk by Leslie Kaelbling at the European Workshop on Reinforcement Learning EWRL, in Utrecht 2001.

Eligibility Traces

In MC methods, the updates are based on the entire sequence of observed rewards until the end of an episode. In TD methods, the estimates are based on the samples of immediate rewards and the next states. An intermediate approach is to use the *n-step-truncated-return* $R_t^{(n)}$, obtained from a whole sequence of returns:

$$R_t^{(n)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^n V_t(s_{t+n})$$

With this, one can go to the approach of computing the updates of values based on several *n*-step returns. The family of $\text{TD}(\lambda)$, with $0 \leq \lambda \leq 1$, combines *n*-step returns weighted proportionally to λ^{n-1} .

The problem with this is that we would have to wait indefinitely to compute $R_t^{(\infty)}$. This view is useful for theoretical analysis and understanding of *n*-step backups. It is called the *forward view of the TD(λ) algorithm*. However, the usual way to implement this kind of updates is called the *backward view of the TD(λ) algorithm* and is done by using *eligibility traces*, which is an incremental implementation of the same idea.

Eligibility traces are a way to perform some kind of *n*-step backups in an elegant way. For each state $s \in S$, an eligibility $e_t(s)$ is kept in memory. They are initialized at 0 and incremented every time according to:

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma \lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$

where λ is the *trace decay parameter*. The trace for each state is increased every time that state is visited and decreases exponentially otherwise. Now δ_t is the *temporal difference* error at stage t :

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

On every step, all states are updated in proportion to their eligibility traces as in:

$$V(s) := V(s) + \alpha \delta_t e_t(s)$$

The forward and backward view on eligibility traces can be proved equivalent Sutton and Barto (1998). For $\lambda = 1$, $\text{TD}(\lambda)$ is essentially the same as MC, because it considers the complete return, and for $\lambda = 0$, $\text{TD}(\lambda)$ uses just the immediate return as in all one-step RL algorithms. Eligibility traces are a general mechanism to learn from *n*-step returns. They can be combined with all of the model-free methods we have described in the previous section. Watkins (1989) combined *Q*-learning with eligibility traces in the $Q(\lambda)$ -algorithm. Peng and Williams (1996) proposed a similar algorithm, and Wiering and Schmidhuber (1998b) and Reynolds (2002) both proposed efficient versions of $Q(\lambda)$. The problem with combining eligibility traces with learning *control* is that special care has to be taken in case of *exploratory*

```

Require: initialize Q and Model arbitrarily
repeat
     $s \in S$  is the start state
     $a := \varepsilon\text{-greedy}(s, Q)$ 
    update  $Q$ 
    update Model
    for  $i := 1$  to  $n$  do
         $s :=$  randomly selected observed state
         $a :=$  random, previously selected action from  $s$ 
        update  $Q$  using the model
    until Sufficient Performance

```

Algorithm 5. Dyna-Q Sutton and Barto (1998)

actions, which can break the intended meaning of the n -step return for the current policy that is followed. In Watkins (1989)'s version, eligibility traces are reset every time an exploratory action is taken. Peng and Williams (1996)'s version is, in that respect more efficient, in that traces do not have to be set to zero every time. SARSA(λ) Sutton and Barto (1998) is more safe in this respect, because action selection is on-policy. Another recent on-policy learning algorithm is the QV(λ) algorithm by Wiering (2005). In QV(λ)-learning two value functions are learned; TD(λ) is used for learning a state value function V and one-step Q -learning is used for learning a state-action value function, based on V .

Learning and Using a Model: Learning and Planning

Even though RL methods can function without a model of the MDP, such a model can be useful to speed up learning, or bias exploration. A learned model can also be useful to do more efficient value *updating*. A general guideline is when experience is costly, it pays off to learn a model. In RL model-learning is usually targeted at the specific learning task defined by the MDP, i.e. determined by the rewards and the goal. In general, learning a model is most often useful because it gives knowledge about the *dynamics* of the environment, such that it can be used for other tasks too (see Drescher (1991) for extensive elaboration on this point).

The DYNA architecture Sutton (1990, 1991b,a); Sutton and Barto (1998) is a simple way to use the model to amplify experiences. Algorithm 5 shows DYNA-Q which combines Q -learning with planning. In a continuous loop, Q -learning is interleaved with series of extra updates using a model that is constantly updated too. DYNA needs less interactions with the environment, because it *replays* experience to do more value updates.

A related method that makes more use of experience using a learned model is *prioritized sweeping* (PS) Moore and Atkeson (1993). Instead of selecting states to be updated randomly (as in DYNA), PS prioritizes updates based on their change in values. Once a state is updated, the PS algorithm considers all states that can reach that state, by looking at the transition model, and sees whether these states will have

to be updated as well. The order of the updates is determined by the size of the value updates. The general mechanism can be summarized as follows. In each step i) one remembers the old value of the current state, ii) one updates the state value with a full backup using the learned model, iii) one sets the priority of the current state to 0, iv) one computes the change δ in value as the result of the backup, v) one uses this difference to modify *predecessors* of the current state (determined by the model); all states leading to the current state get a priority update of $\delta \times T$, where T is the probability a successor state leads to the current state that is updated. The number of value backups is a parameter to be set in the algorithm. Overall, PS focuses the backups to where they are expected to most quickly reduce the error. Another example of using planning in model-based RL is Wiering (2002).

1.8 Conclusions

This chapter has provided the necessary background of Markov decision processes, dynamic programming and reinforcement learning. Core elements of many solution algorithms which will be discussed in subsequent chapters are Bellman equations, value updates, exploration and sampling.

References

- Bain, M., Sammut, C.: A framework for behavioral cloning. In: Muggleton, S.H., Furukawa, K., Michie, D. (eds.) *Machine Intelligence*, vol. 15, pp. 103–129. Oxford University Press (1995)
- Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13, 835–846 (1983)
- Barto, A.G., Bradtke, S.J., Singh, S.: Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1), 81–138 (1995)
- Bellman, R.E.: *Dynamic Programming*. Princeton University Press, Princeton (1957)
- Bertsekas, D.P.: *Dynamic Programming and Optimal Control*, vol. 1, 2. Athena Scientific, Belmont (1995)
- Bertsekas, D.P., Tsitsiklis, J.: *Neuro-Dynamic Programming*. Athena Scientific, Belmont (1996)
- Bonet, B., Geffner, H.: Faster heuristic search algorithms for planning with uncertainty and full feedback. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1233–1238 (2003a)
- Bonet, B., Geffner, H.: Labeled RTDP: Improving the convergence of real-time dynamic programming. In: *Proceedings of the International Conference on Artificial Intelligence Planning Systems (ICAPS)*, pp. 12–21 (2003b)
- Boutilier, C.: Knowledge Representation for Stochastic Decision Processes. In: Veloso, M.M., Wooldridge, M.J. (eds.) *Artificial Intelligence Today. LNCS (LNAI)*, vol. 1600, pp. 111–152. Springer, Heidelberg (1999)

- Boutilier, C., Dean, T., Hanks, S.: Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11, 1–94 (1999)
- Brafman, R.I., Tennenholtz, M.: R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research (JMLR)* 3, 213–231 (2002)
- Dean, T., Kaelbling, L.P., Kirman, J., Nicholson, A.: Planning under time constraints in stochastic domains. *Artificial Intelligence* 76, 35–74 (1995)
- Dixon, K.R., Malak, M.J., Khosla, P.K.: Incorporating prior knowledge and previously learned information into reinforcement learning agents. Tech. rep., Institute for Complex Engineered Systems, Carnegie Mellon University (2000)
- Dorigo, M., Colombetti, M.: Robot Shaping: An Experiment in Behavior Engineering. The MIT Press, Cambridge (1997)
- Drescher, G.: Made-Up Minds: A Constructivist Approach to Artificial Intelligence. The MIT Press, Cambridge (1991)
- Ferguson, D., Stentz, A.: Focussed dynamic programming: Extensive comparative results. Tech. Rep. CMU-RI-TR-04-13, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania (2004)
- Främling, K.: Bi-memory model for guiding exploration by pre-existing knowledge. In: Driessens, K., Fern, A., van Otterlo, M. (eds.) *Proceedings of the ICML-2005 Workshop on Rich Representations for Reinforcement Learning*, pp. 21–26 (2005)
- Großmann, A.: Adaptive state-space quantisation and multi-task reinforcement learning using constructive neural networks. In: *From Animals to Animats: Proceedings of The International Conference on Simulation of Adaptive Behavior (SAB)*, pp. 160–169 (2000)
- Hansen, E.A., Zilberstein, S.: LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129, 35–62 (2001)
- Howard, R.A.: Dynamic Programming and Markov Processes. The MIT Press, Cambridge (1960)
- Kaelbling, L.P.: Learning in Embedded Systems. The MIT Press, Cambridge (1993)
- Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4, 237–285 (1996)
- Kearns, M., Singh, S.: Near-optimal reinforcement learning in polynomial time. In: *Proceedings of the International Conference on Machine Learning (ICML)* (1998)
- Koenig, S., Liu, Y.: The interaction of representations and planning objectives for decision-theoretic planning. *Journal of Experimental and Theoretical Artificial Intelligence* 14(4), 303–326 (2002)
- Konda, V., Tsitsiklis, J.: Actor-critic algorithms. *SIAM Journal on Control and Optimization* 42(4), 1143–1166 (2003)
- Konidaris, G.: A framework for transfer in reinforcement learning. In: *ICML-2006 Workshop on Structural Knowledge Transfer for Machine Learning* (2006)
- Kushmerick, N., Hanks, S., Weld, D.S.: An algorithm for probabilistic planning. *Artificial Intelligence* 76(1-2), 239–286 (1995)
- Littman, M.L., Dean, T., Kaelbling, L.P.: On the complexity of solving Markov decision problems. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 394–402 (1995)
- Mahadevan, S.: Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning* 22, 159–195 (1996)
- Maloof, M.A.: Incremental rule learning with partial instance memory for changing concepts. In: *Proceedings of the International Joint Conference on Neural Networks*, pp. 2764–2769 (2003)

- Mataric, M.J.: Reward functions for accelerated learning. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 181–189 (1994)
- Matthews, W.H.: Mazes and Labyrinths: A General Account of their History and Developments. Longmans, Green and Co., London (1922); Mazes & Labyrinths: Their History & Development. Dover Publications, New York (reprinted in 1970)
- McMahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 569–576 (2005)
- Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13(1), 103–130 (1993)
- Ng, A.Y., Harada, D., Russell, S.J.: Policy invariance under reward transformations: Theory and application to reward shaping. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 278–287 (1999)
- Peng, J., Williams, R.J.: Incremental multi-step Q-learning. *Machine Learning* 22, 283–290 (1996)
- Puterman, M.L.: *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York (1994)
- Puterman, M.L., Shin, M.C.: Modified policy iteration algorithms for discounted Markov decision processes. *Management Science* 24, 1127–1137 (1978)
- Ratitch, B.: On characteristics of Markov decision processes and reinforcement learning in large domains. PhD thesis, The School of Computer Science, McGill University, Montreal (2005)
- Reynolds, S.I.: Reinforcement learning with exploration. PhD thesis, The School of Computer Science, The University of Birmingham, UK (2002)
- Rummery, G.A.: Problem solving with reinforcement learning. PhD thesis, Cambridge University, Engineering Department, Cambridge, England (1995)
- Rummery, G.A., Niranjan, M.: On-line Q-Learning using connectionist systems. Tech. Rep. CUED/F-INFENG/TR 166, Cambridge University, Engineering Department (1994)
- Russell, S.J., Norvig, P.: *Artificial Intelligence: a Modern Approach*, 2nd edn. Prentice Hall, New Jersey (2003)
- Schaeffer, J., Plaat, A.: Kasparov versus deep blue: The re-match. *International Computer Chess Association Journal* 20(2), 95–101 (1997)
- Schwartz, A.: A reinforcement learning method for maximizing undiscounted rewards. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 298–305 (1993)
- Singh, S., Jaakkola, T., Littman, M., Szepesvari, C.: Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning* 38(3), 287–308 (2000)
- Sutton, R.S.: Learning to predict by the methods of temporal differences. *Machine Learning* 3, 9–44 (1988)
- Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 216–224 (1990)
- Sutton, R.S.: DYNA, an integrated architecture for learning, planning and reacting. In: Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures, pp. 151–155 (1991a)
- Sutton, R.S.: Reinforcement learning architectures for animats. In: From Animals to Animats: Proceedings of The International Conference on Simulation of Adaptive Behavior (SAB), pp. 288–296 (1991b)

- Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Proceedings of the Neural Information Processing Conference (NIPS), pp. 1038–1044 (1996)
- Sutton, R.S., Barto, A.G.: Reinforcement Learning: an Introduction. The MIT Press, Cambridge (1998)
- Tash, J., Russell, S.J.: Control strategies for a stochastic planner. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 1079–1085 (1994)
- Watkins, C.J.C.H.: Learning from delayed rewards. PhD thesis, King's College, Cambridge, England (1989)
- Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning 8(3/4) (1992); Special Issue on Reinforcement Learning
- Wiering, M.A.: Explorations in efficient reinforcement learning. PhD thesis, Faculteit der Wiskunde, Informatica, Natuurkunde en Sterrenkunde, Universiteit van Amsterdam (1999)
- Wiering, M.A.: Model-based reinforcement learning in dynamic environments. Tech. Rep. UU-CS-2002-029, Institute of Information and Computing Sciences, University of Utrecht, The Netherlands (2002)
- Wiering, M.A.: QV(λ)-Learning: A new on-policy reinforcement learning algorithm. In: Proceedings of the 7th European Workshop on Reinforcement Learning (2005)
- Wiering, M.A., Schmidhuber, J.H.: Efficient model-based exploration. In: From Animals to Animats: Proceedings of The International Conference on Simulation of Adaptive Behavior (SAB), pp. 223–228 (1998a)
- Wiering, M.A., Schmidhuber, J.H.: Fast online Q(λ). Machine Learning 33(1), 105–115 (1998b)
- Winston, W.L.: Operations Research Applications and Algorithms, 2nd edn. Thomson Information/Publishing Group, Boston (1991)
- Witten, I.H.: An adaptive optimal controller for discrete-time markov environments. Information and Control 34, 286–295 (1977)

Part II

Efficient Solution Frameworks

Chapter 2

Batch Reinforcement Learning

Sascha Lange, Thomas Gabel, and Martin Riedmiller

Abstract. Batch reinforcement learning is a subfield of dynamic programming-based reinforcement learning. Originally defined as the task of learning the best possible policy from a fixed set of a priori-known transition samples, the (batch) algorithms developed in this field can be easily adapted to the classical online case, where the agent interacts with the environment while learning. Due to the efficient use of collected data and the stability of the learning process, this research area has attracted a lot of attention recently. In this chapter, we introduce the basic principles and the theory behind batch reinforcement learning, describe the most important algorithms, exemplarily discuss ongoing research within this field, and briefly survey real-world applications of batch reinforcement learning.

2.1 Introduction

Batch reinforcement learning is a subfield of dynamic programming (DP) based reinforcement learning (RL) that has vastly grown in importance during the last years. Historically, the term ‘batch RL’ is used to describe a reinforcement learning setting, where the complete amount of learning experience—usually a set of transitions sampled from the system—is fixed and given a priori (Ernst et al, 2005a). The task of the learning system then is to derive a solution—usually an optimal policy—out of this given batch of samples.

In the following, we will relax this assumption of an a priori fixed set of training experience. The crucial benefit of batch algorithms lies in the way they handle a batch of transitions and get the best out of it, rather than in the fact that this set is fixed. From this perspective, batch RL algorithms are characterized by two basic

Sascha Lange · Thomas Gabel · Martin Riedmiller
Albert-Ludwigs-Universität Freiburg, Faculty of Engineering, Georges-Köhler-Allee 079,
D-79110 Freiburg, Germany
e-mail: {slange, tgabel, riedmiller}@informatik.uni-freiburg.de

constituents: all observed transitions are stored and updates occur synchronously on the whole batch of transitions ('fitting'). In particular, this allows for the definition of 'growing batch' methods, that are allowed to extend the set of sample experience in order to incrementally improve their solution. From the interaction perspective, the growing batch approach minimizes the difference between batch methods and pure online learning methods.

The benefits that come with the batch idea—namely, stability and data-efficiency of the learning process—account for the large interest in batch algorithms. Whereas basic algorithms like Q-learning usually need many interactions until convergence to good policies, thus often rendering a direct application to real applications impossible, methods including ideas from batch reinforcement learning usually converge in a fraction of the time. A number of successful examples of applying ideas originating from batch RL to learning in the interaction with real-world systems have recently been published (see sections 2.6.2 and 2.6.5).

In this chapter, we will first define the batch reinforcement learning problem and its variants, which form the problem space treated by batch RL methods. We will then give a brief historical recap of the development of the central ideas that, in retrospect, built the foundation of all modern batch RL algorithms. On the basis of the problem definition and the introduced ideas, we will present the most important algorithms in batch RL. We will discuss their theoretical properties as well as some variations that have a high relevance for practical applications. This includes a treatment of Neural Fitted Q Iteration (NFQ) and some of its applications, as it has proven a powerful tool for learning on real systems. With the application of batch methods to both visual learning of control policies and solving distributed scheduling problems, we will briefly discuss on-going research.

2.2 The Batch Reinforcement Learning Problem

Batch reinforcement learning was historically defined as the class of algorithms developed for solving a particular learning problem—namely, the batch reinforcement learning problem.

2.2.1 The Batch Learning Problem

As in the general reinforcement learning problem defined by Sutton and Barto (1998), the task in the batch learning problem is to find a policy that maximizes the sum of expected rewards in the familiar agent-environment loop. However, differing from the general case, in the batch learning problem the agent itself is not allowed to interact with the system during learning. Instead of observing a state s ,

trying an action a and adapting its policy according to the subsequent following state s' and reward r , the learner only receives a set $\mathcal{F} = \{(s_t, a_t, r_{t+1}, s_{t+1}) | t = 1, \dots, p\}$ of p transitions (s, a, r, s') sampled from the environment¹.

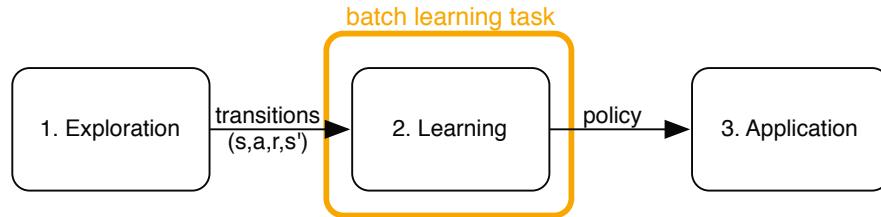


Fig. 2.1 The three distinct phases of the batch reinforcement learning process: 1: Collecting transitions with an arbitrary sampling strategy. 2: Application of (batch) reinforcement learning algorithms in order to learn the best possible policy from the set of transitions. 3: Application of the learned policy. Exploration is not part of the batch learning task. During the application phase, that isn't part of the learning task either, policies stay fixed and are not improved further.

In the most general case of this batch reinforcement learning problem the learner cannot make any assumptions on the sampling procedure of the transitions. They may be sampled by an arbitrary—even purely random—policy; they are not necessarily sampled uniformly from the state-action space $S \times A$; they need not even be sampled along connected trajectories. Using only this information, the learner has to come up with a policy that will then be used by the agent to interact with the environment. During this application phase the policy is fixed and not further improved as new observations come in. Since the learner itself is not allowed to interact with the environment, and the given set of transitions is usually finite, the learner cannot be expected to always come up with an optimal policy. The objective has therefore been changed from learning an optimal policy—as in the general reinforcement learning case—to deriving the best possible policy from the given data.

The distinct separation of the whole procedure into three phases—exploring the environment and collecting state transitions and rewards, learning a policy, and application of the learned policy—their sequential nature, and the data passed at the interfaces is further clarified in figure 2.1. Obviously, treatment of the exploration-exploitation dilemma is not subject to algorithms solving such a pure batch learning problem, as the exploration is not part of the learning task at all.

¹ The methods presented in this chapter all assume a markovian state representation and the transitions \mathcal{F} to be sampled from a discrete-time Markov decision process (MDP, see chapter 1). For a treatment of only partially observable decision processes see chapter 12.

2.2.2 *The Growing Batch Learning Problem*

Although this batch reinforcement learning problem historically has been the start of the development of batch reinforcement learning algorithms, modern batch RL algorithms are seldom used in this ‘pure’ batch learning problem. In practice, exploration has an important impact on the quality of the policies that can be learned. Obviously, the distribution of transitions in the provided batch must resemble the ‘true’ transition probabilities of the system in order to allow the derivation of good policies. The easiest way to achieve this is to sample the training examples from the system itself, by simply interacting with it. But when sampling from the real system, another aspect becomes important: the covering of the state space by the transitions used for learning. If ‘important’ regions—e.g. states close to the goal state—are not covered by any samples, then it is obviously not possible to learn a good policy from the data, since important information is missing. This is a real problem because in practice, a completely ‘uninformed’ policy—e.g. a purely random policy—is often not able to achieve an adequate covering of the state space—especially in the case of attractive starting states and hard to reach desirable states. It is often necessary to already have a rough idea of a good policy in order to be able to explore interesting regions that are not in the direct vicinity of the starting states.

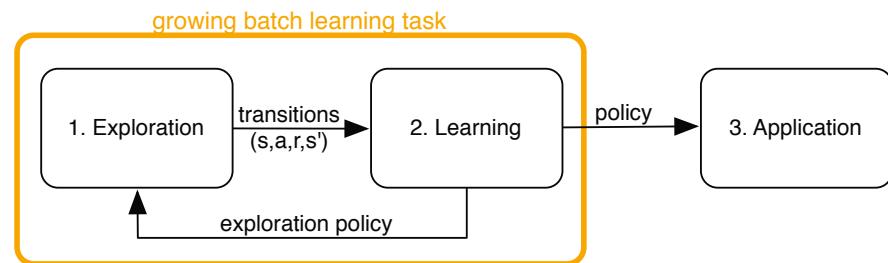


Fig. 2.2 The growing batch reinforcement learning process has the same three phases as the ‘pure’ batch learning process depicted in figure 2.1. But differing from the pure batch process, the growing batch learning process alternates for several times between the exploration and the learning phase, thus incrementally ‘growing’ the batch of stored transitions using intermediate policies.

This is the main reason why a third variant of the reinforcement learning problem became a popular practice, somehow positioned in between the pure online problem and the pure batch problem. Since the main idea of this third type of learning problem is to alternate between phases of exploration, where a set of training examples is grown by interacting with the system, and phases of learning, where the whole batch of observations is used (see fig. 2.2), we will refer to it as the ‘growing batch’ learning problem. In the literature, this growing batch approach can be found in several different guises; the number of alternations between episodes of

exploration and episodes of learning can be in the whole range of being as close to the pure batch approach as using only two iterations (Riedmiller et al, 2008) to recalculating the policy after every few interactions—e.g. after finishing one episode in a shortest-path problem (Kalyanakrishnan and Stone, 2007; Lange and Riedmiller, 2010a). In practice, the growing batch approach is the modeling of choice when applying batch reinforcement learning algorithms to real systems. Since from the interaction perspective the growing batch approach is very similar to the ‘pure’ online approach—the agent improves its policy *while* interacting with the system—the interaction perspective, with the distinction between ‘online’ and ‘offline’, isn’t that useful anymore for identifying batch RL algorithms. When talking about ‘batch’ RL algorithms now, it’s more important to look at the algorithms and search for typical properties of the specific update rules.

2.3 Foundations of Batch RL Algorithms

Model-free online learning methods like Q-learning are appealing from a conceptual point of view and have been very successful when applied to problems with small, discrete state spaces. But when it comes to applying them to more realistic systems with larger and, possibly, continuous state spaces, these algorithms come up against limiting factors. In this respect, there can be identified three independent problems:

1. the ‘exploration overhead’, causing slow learning in practice
2. inefficiencies due to the stochastic approximation
3. stability issues when using function approximation

A common factor in modern batch RL algorithms is that these algorithms typically address all three issues and come up with specific solutions to each of them. In the following, we will discuss these problems in more detail and present the proposed solutions (in historical order) that now form the defining ideas behind modern batch reinforcement learning.

The Idea of ‘Experience Replay’ for Addressing the Exploration Overhead

In order to explain the problem referred to as ‘exploration overhead’, let us for a moment consider the common Q-update rule for model-free online learning with learning rate α and discount factor γ as given by

$$Q'(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right] \quad (2.1)$$

(see chapter 1). In pure online Q-learning the agent alternates between learning and exploring with practically every single time step: in state s the agent selects and executes an action a , and then, when observing the subsequent state s' and reward r , it immediately updates the value function (and thus the corresponding greedy

policy; see chapter 1) according to (2.1), afterwards forgetting the ‘experienced’ state transition tuple (s,a,r,s') . It then returns to exploring with the updated policy $Q'(s,a)$. Although this approach is guaranteed to converge in the limit, there is a severe performance problem with these ‘local’ updates along actually experienced transitions. When, for example, updating the Q-value of state-action pair (s_t,a_t) in time step t this may influence the values (s_{t-1},a) for all $a \in A$ of a preceding state s_{t-1} . However, this change will not back-propagate immediately to all the involved preceding states, as the states preceding s_t are only updated the next time they are visited. And states preceding those preceding states s_{t-1} need yet another trial afterwards to be adapted to the new value of s_{t-1} . The performance problem with this is that these interactions are not actually needed to collect more information from the system in the first place, but mainly are needed to spread already available information through the whole state space in reverse order along trajectories. In practice, many interactions in Q-learning are of this ‘spreading’ type of interaction. This problem becomes even more pronounced when considering that those updates in model-free Q-learning are ‘physical’—in the sense of needing real interaction with the system—and can seldomly be sampled in ordered sweeps across the whole state space or, at least, according to a uniform distribution.

In order to overcome this performance issue, Lin (1992) introduced the idea of ‘experience replay’. Although the intention was to solve the ‘exploration overhead’ problem in online learning, in retrospect, we might identify it as basic—but nevertheless first—technique used for addressing the growing batch problem. The idea behind experience replay is to speed up convergence not only by using observed state transitions (the experience) once, but replaying them repeatedly to the agent as if they were new observations collected while interacting with the system. In fact, one can store a few to as many as all transitions observed up until that point and use them to update the Q-function for every single stored transition according to (2.1) after every interaction with the system. It is exactly the information in the stored transitions that is missing in basic online Q-learning to be able to back-propagate information from updated states to preceding states without further interaction. The transitions collected when using experience replay resemble the connection between individual states and make more efficient use of this information, by spreading it along these connections, and, ideally, speeding up convergence.

The Idea of ‘Fitting’ to Address Stability Issues

In online RL it is common to use ‘asynchronous’ updates in the sense that the value function after each observation is immediately updated locally for that particular state, leaving all other states untouched. In the discrete case, this means updating one single Q-value for a state-action pair (s,a) in Q-learning—thereby immediately ‘over-writing’ the value of the starting state of the transition. Subsequent updates would use this updated value for their own update. This idea was also used with function approximation, first performing a DP update like, for example,

$$\bar{q}_{s,a} = r + \gamma \max_{a' \in A} f(s', a') , \quad (2.2)$$

recalculating the approximated value $\bar{q}_{s,a}$ of the present state action pair (s,a) by, e.g., adding immediate reward and approximated value of the subsequent state, and then immediately ‘storing’ the new value in the function approximator in the sense of moving the value of the approximation slightly towards the value of the new estimate $\bar{q}_{s,a}$ for the state-action pair (s,a) :

$$f'(s,a) \leftarrow (1 - \alpha) f(s,a) + \alpha \bar{q}_{s,a} . \quad (2.3)$$

Please note that (2.2) and (2.3) are just re-arranged forms of equation (2.1), using an arbitrary function approximation scheme for calculating an approximation f' of the ‘true’ Q-value function Q' .

Baird (1995), Gordon (1996) and others have shown examples where particular combinations of Q-learning and similar updates with function approximators behave instably or even lead to sure divergence. Stable behavior can be proven only for particular instances of combinations of approximation schemes and update rules, or under particular circumstances and assumptions on the system and reward structure (Schoknecht and Merke, 2003). In practice it required extensive experience on behalf of the engineer, in order to get a particular learning algorithm to work on a system. The observed stability issues are related to the interdependency of errors made in function approximation and deviations of the estimated value function from the optimal value function. Whereas the DP update (2.2) tries to gradually decrease the difference between $Q(s,a)$ and the optimal Q-function $Q^*(s,a)$, storing the updated value in the function approximator in step (2.3) might (re-)introduce an even larger error. Moreover, this approximation error influences all subsequent DP updates and may work against the contraction or even prevent it. The problem becomes even worse when global function approximators—like, for example, multi-layer perceptrons (Rumelhart et al, 1986; Werbos, 1974)—are used; improving a single Q-value of a state-action pair might impair all other approximations throughout the entire state space.

In this situation Gordon (1995a) came up with the compelling idea of slightly modifying the update scheme in order to separate the dynamic programming step from the function approximation step. The idea is to first apply a DP update to all members of a set of so-called ‘supports’ (points distributed throughout the state space) calculate new ‘target values’ for all supports (like in (2.2)), and then use supervised learning to train (‘fit’) a function approximator on all these new target values, thereby replacing the local updates of (2.3) (Gordon, 1995a). In this sense, the estimated Q-function is updated ‘synchronously’, with updates occurring at all supports at the same time. Although Gordon introduced this fitting idea within the setting of model-based value iteration, it became the foundation, and perhaps even the starting point, of all modern batch algorithms.

Replacing Inefficient Stochastic Approximation

Gordon discussed the possibility of transferring the ‘fitting’ idea to model-free—sample-based—approaches like, e.g., Q-learning, but did not find a solution and identified several convergence issues when estimating the values at the supports from samples in their surrounding. Ormoneit and Sen finally came up with the solution of how to adapt and apply his idea to the sample-based case. In their work, Ormoneit and Sen (2002) propose not to use arbitrarily selected supports in the state space to approximate value functions, but rather to use the sampled transitions directly to approximate the value function—either at the starting or at the ending states of the transitions—with the help of a kernel-based approximator. Their idea is to calculate an estimate of the value of each explored state-action pair under the actually observed transition (reward plus expected value of subsequent state), and then estimate the values of the subsequent states by averaging over the values of nearby transitions. Technically, the trick is to replace the exact DP-operator that was introduced by Gordon with a non-exact random operator (see section 2.4.1 for a formal definition). This random operator does not use the exact models in the DP-step. Instead, it only estimates the exact DP-operation by using a random sampling of transitions drawn from the unknown transition models. Lagoudakis and Parr (2001, 2003) came up with a similar idea independently.

As a side effect, calculating the costs of particular transitions and estimating the values of states (and actions) by averaging over the stored samples solved another performance problem connected to the stochastic approximation in regular Q-learning. Whereas in model-based value iteration the value of a state according to the current values of its possible subsequent states can be updated in a single step using the transition model, in Q-learning—due to the replacement of the model by stochastic approximation—such an update requires many visits to the state in question. Moreover, since we are using one learning rate for the entire state space, this learning rate cannot be adjusted in a way that is optimal for all states depending on the number of visits. In practice, this makes reaching the optimal convergence rate impossible. The algorithm of Ormoneit and Sen does not rely on stochastic approximation, but rather implicitly estimates the transition model by averaging over the observed transitions that—if collected in interaction with the system—actually form a random sampling from the true distributions.

2.4 Batch RL Algorithms

In their work, Ormoneit and Sen proposed a ‘unified’ kernel-based algorithm that could actually be seen as a general framework upon which several later algorithms are based. Their ‘kernel-based approximate dynamic programming’ (KADP) brought together the ideas of experience replay (storing and re-using experience), fitting (separation of DP-operator and approximation), and kernel-based self-approximation (sample-based).

2.4.1 Kernel-Based Approximate Dynamic Programming

Ormoneit's kernel-based approximate dynamic programming solves an approximated version of the 'exact' Bellman-equation

$$V = HV$$

that is expressed in

$$\hat{V} = \hat{H}\hat{V}.$$

It not only uses an approximation \hat{V} of the 'true' state-value function V —as Gordon's fitted value iteration did—but also uses an approximate version \hat{H} of the exact DP-operator H itself.

The KADP algorithm for solving this equation works as follows: starting from an arbitrary initial approximation \hat{V}^0 of the state-value function, each iteration i of the KADP-algorithm consists of solving the equation

$$\hat{V}^{i+1} = H_{max}\hat{H}_{dp}^a\hat{V}^i$$

for a given set

$$\mathcal{F} = \{(s_t, a_t, r_{t+1}, s_{t+1}) | t = 1, \dots, p\}$$

of p transitions (s, a, r, s') . In this equation, the approximate DP-operator

$$\hat{H} = H_{max}\hat{H}_{dp}^a$$

has been split into an exact part H_{max} maximizing over actions and an approximate random operator \hat{H}_{dp}^a approximating the 'true' (model-based) DP-step for individual actions from the observed transitions. The first half of this equation is calculated according to the sample-based DP update

$$\hat{Q}_a^{i+1}(\sigma) := \hat{H}_{dp}^a\hat{V}^i(\sigma) = \sum_{(s, a, r, s') \in \mathcal{F}_a} k(s, \sigma) [r + \gamma\hat{V}^i(s')] . \quad (2.4)$$

Using a weighting-kernel $k(\cdot, \sigma)$, this equation calculates a weighted average of the well-known Q-updates

$$r + \gamma\hat{V}^i(s') = r + \gamma \max_{a' \in A} \hat{Q}^i(s', a'),$$

(please note the similarity to equation (2)) along all transitions $(s, a, r, s') \in \mathcal{F}_a$, where $\mathcal{F}_a \subset \mathcal{F}$ is the subset of \mathcal{F} that contains only the transitions $(s, a, r, s') \in \mathcal{F}$ that used the particular action a . The weighting kernel is chosen in such a way that more distant samples have a smaller influence on the resulting sum than closer (= more similar) samples.

The second half of the equation applies the maximizing-operator H_{max} to the approximated Q-functions \hat{Q}_a^{i+1} :

$$\hat{V}^{i+1}(s) = H_{\max} \hat{Q}_a^{i+1}(s) = \max_{a \in A} \hat{Q}_a^{i+1}(s). \quad (2.5)$$

Please note that this algorithm uses an individual approximation $\hat{Q}_a^{i+1} : S \mapsto R$ for each action $a \in A$ in order to approximate the Q-function $Q_a^{i+1} : S \times A \mapsto R$. Furthermore, a little bit counter-intuitively, in a practical implementation, this last equation is actually evaluated and stored for all ending states s' of all transitions $(s, a, r, s') \in \mathcal{F}$ —not the starting states s . This decision to store the ending states is explained by noting that in the right-hand side of equation (4) we only query the present estimate of the value function at the ending states of the transitions—never its starting states (see Ormoneit and Sen (2002), section 4). This becomes clearer in figure 2.3.

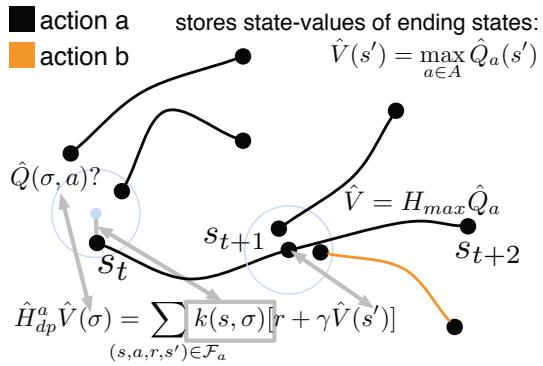


Fig. 2.3 Visualization of the kernel-based approximation in KADP. For computing the Q-value $\hat{Q}(\sigma, a) = \hat{H}_{dp}^a(\sigma)\hat{V}(\sigma)$ of (σ, a) at an arbitrary state $\sigma \in S$, KADP uses the starting states s of nearby transitions (s, a, r, s') only for calculating the weighting factors $k(s, \sigma)$, however, depends on the state values $\hat{V}(s')$ of ending states s' (in the depicted example $s = s_t$ and $s' = s_{t+1}$).

Iterating over equations (2.4) and (2.5), calculating a sequence of approximations \hat{V}^{i+1} with $i = 0, 1, 2, \dots$, the algorithm will ultimately converge to a unique solution \hat{V} when explicitly storing the values $\hat{V}^i(s')$ of \hat{V}^i at the end-points s' of the transitions (s, a, r, s') and using weighting-kernels that adhere to the ‘averager’ restriction from Gordon (1995a). For meeting this restriction the weights need to, A), add up to one

$$\sum_{(s, a, r, s') \in \mathcal{F}_a} k(s, \sigma) = 1 \quad \forall \sigma \in S$$

and, B), to be non-negative

$$k(s, \sigma) \geq 0 \quad \forall \sigma \in S \quad \forall (s, a, r, s') \in \mathcal{F}_a.$$

The decision to store and iterate over state-values instead of state-action values results in the necessity of applying another DP-step in order to derive a greedy policy from the result of the KADP algorithm:

$$\begin{aligned}\pi^{i+1}(\sigma) &= \arg \max_{a \in A} \hat{H}_{dp}^a \hat{V}^i(\sigma) \\ &= \arg \max_{a \in A} \sum_{(s,a,r,s') \in \mathcal{F}_a} k(s,\sigma) [r + \gamma \hat{V}^i(s')] .\end{aligned}$$

This might appear to be a small problem from an efficiency point of view, as you have more complex computations as in Q-learning and need to remember all transitions in the application phase, but it is not a theoretical problem. Applying the DP-operator to the fixed point of $\hat{V} = \hat{H}\hat{V}$ does not change anything but results in the same unique fixed point.

2.4.2 Fitted Q Iteration

Perhaps the most popular algorithm in batch RL is Damien Ernst's 'Fitted Q Iteration' (FQI, Ernst et al (2005a)). It can be seen as the 'Q-Learning of batch RL', as it is actually a straight-forward transfer of the basic Q-learning update-rule to the batch case. Given a fixed set $\mathcal{F} = \{(s_t, a_t, r_{t+1}, s_{t+1}) | t = 1, \dots, p\}$ of p transitions (s, a, r, s') and an initial Q-value \bar{q}^0 (Ernst et al (2005a) used $\bar{q}^0 = 0$) the algorithm starts by initializing an initial approximation \hat{Q}^0 of the Q-function Q^0 with $\hat{Q}^0(s, a) = \bar{q}^0$ for all $(s, a) \in S \times A$. It then iterates over the following two steps:

1. Start with an empty set P^{i+1} of patterns $(s, a; \bar{q}_{s,a}^{i+1})$. Then, for each transition $(s, a, r, s') \in \mathcal{F}$ calculate a new target Q-value $\bar{q}_{s,a}^{i+1}$ according to

$$\bar{q}_{s,a}^{i+1} = r + \gamma \max_{a' \in A} \hat{Q}^i(s', a') \quad (2.6)$$

(similar to the update in equation (2.2)) and add a corresponding pattern $(s, a; \bar{q}_{s,a}^{i+1})$ to the pattern set; thus:

$$P^{i+1} \leftarrow P^{i+1} \cup \{(s, a; \bar{q}_{s,a}^{i+1})\} .$$

2. Use supervised learning to train a function approximator on the pattern set P^{i+1} . The resulting function \hat{Q}^{i+1} then is an approximation of the Q-function Q^{i+1} after $i + 1$ steps of dynamic programming.

Originally, Ernst proposed randomized trees for approximating the value function. After fixing their structure, these trees can also be represented as kernel-based averagers, thereby reducing step 2 to

$$\hat{Q}_a^{i+1}(\sigma) = \sum_{(s, a; \bar{q}_{s,a}^{i+1}) \in P_a^{i+1}} k(s, \sigma) \bar{q}_{s,a}^{i+1} , \quad (2.7)$$

with the weights $k(\cdot, \sigma)$ determined by the structure of the tree. This variant of FQI constructs an individual approximation \hat{Q}_a^{i+1} for each discrete action $a \in A$ which, together, form the approximation $\hat{Q}^{i+1}(s, a) = \hat{Q}_a^{i+1}(s)$ (Ernst et al, 2005a, section 3.4). Besides this variant of FQI, Ernst also proposed a variant with continuous actions. We may refer the interested reader to Ernst et al (2005a) for a detailed description of this.

From a theoretical stand-point, Fitted Q Iteration is nevertheless based on Ormoneit and Sen's theoretical framework. The similarity between Fitted Q Iteration and KADP becomes obvious when rearranging equations (2.6) and (2.7):

$$\hat{Q}^{i+1}(\sigma, a) = \hat{Q}_a^{i+1}(\sigma) = \sum_{(s, a; \bar{q}) \in P_a^{i+1}} k(s, \sigma) \bar{q}_{s, a}^{i+1} \quad (2.8)$$

$$= \sum_{(s, a, r, s') \in \mathcal{F}_a} k(s, \sigma) \left[r + \gamma \max_{a' \in A} \hat{Q}_{a'}^i(s') \right]. \quad (2.9)$$

Equation (2.8) is the original averaging step from equation (2.7) in FQI for discrete actions. By inserting FQI's DP-step (2.6) immediately follows (2.9). This result (2.9) is practically identical to the update used in KADP, as can be seen by inserting (2.5) into (2.4):

$$\begin{aligned} \hat{Q}_a^{i+1}(\sigma) &= \sum_{(s, a, r, s') \in \mathcal{F}_a} k(s, \sigma) [r + \gamma \hat{V}^i(s')] \\ &= \sum_{(s, a, r, s') \in \mathcal{F}_a} k(s, \sigma) \left[r + \gamma \max_{a' \in A} \hat{Q}_{a'}^i(s') \right]. \end{aligned}$$

Besides the optional treatment of continuous actions, another difference between FQI and KADP is in the splitting of operators and choice of explicitly represented values. Where KADP explicitly represents and uses state-values in the DP-step (4), FQI explicitly represents the Q-function and calculates state-values $\hat{V}^i(s) = \max_{a \in A} \hat{Q}^i(s, a)$ on the fly by maximizing over actions in its DP-step (2.6). Although ‘lazy-learning’ with kernel-based averaging—as proposed as the standard in KADP—is also allowed in FQI, Ernst assumes the usage of a trained averager or other parametric function approximator for explicitly storing the Q-function. The ‘structure’ of this approximator is neither necessarily related to the starting points nor ending points of transitions. And, since FQI represents the Q-function explicitly, deriving a greedy policy in FQI is rather simple:

$$\pi^i(s) = \arg \max_{a \in A} \hat{Q}^i(s, a)$$

where \hat{Q}^i is realized explicitly, either by a single function approximator (continuous actions) or by a set of function approximators \hat{Q}_a^i for the actions $a \in A$.

These subtle differences in otherwise equivalent algorithms make FQI both, more intuitive and more similar to the online approach of Q-learning. This may account for this algorithm’s greater popularity.

2.4.3 Least-Squares Policy Iteration

Least-squares policy iteration (LSPI, Lagoudakis and Parr (2003)) is another early example of a batch mode reinforcement learning algorithm. In contrast to the other algorithms reviewed in this section, LSPI explicitly embeds the task of solving control problems into the framework of policy iteration (Sutton and Barto, 1998), thus alternating between policy evaluation and policy improvement steps. However, LSPI never stores a policy explicitly. Instead, it works solely on the basis of a state-action value function Q from which a greedy policy is to be derived via $\pi(s) = \arg \max_{a \in A} Q(s, a)$. For the purpose of representing state-action value functions, LSPI employs a parametric linear approximation architecture with a fixed set of k pre-defined basis functions $\phi_i : S \times A$ and a weight vector $w = (w_1, \dots, w_k)^T$. Therefore, any approximated state-action value function \hat{Q} within the scope of LSPI takes the form

$$\hat{Q}(s, a; w) = \sum_{j=1}^k \phi_j(s, a) w_j = \Phi w^T.$$

Its policy evaluation step employs a least-squares temporal difference learning algorithm for the state-action value function (LSQ, Lagoudakis and Parr (2001), later called LSTDQ, Lagoudakis and Parr (2003)). This algorithm takes as input the current policy π_m —as pointed out above, represented by a set of weights that determine a value function \hat{Q} from which π_m is to be derived greedily—as well as a finite set \mathcal{F} of transitions (s, a, r, s') . From these inputs, LSTDQ derives analytically the state-action value function \hat{Q}^{π_m} for the given policy under the state distribution determined by the transition set. Clearly, the derived value function returned \hat{Q}^{π_m} by LSTDQ is, again, fully described by a weight vector w^{π_m} given the above-mentioned linear architecture used.

Generally, the searched for value function is a fixed point of the \hat{H}_π operator

$$(\hat{H}_\pi Q)(s, a) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s, \pi(s), s') \max_{b \in A} Q(s, b), \quad (2.10)$$

i.e. $\hat{H}_{\pi_m} Q^\pi = Q^{\pi_m}$. Thus, a good approximation \hat{Q}^{π_m} should comply to $\hat{H}_{\pi_m} \hat{Q}^{\pi_m} \approx \hat{Q}^{\pi_m} = \Phi(w^{\pi_m})^T$. Practically speaking, LSTDQ aims at finding a vector w^π such that the approximation of the result of applying the \hat{H}_{π_m} operator to \hat{Q}_{π_m} is as near as possible to the true result (in an L_2 norm minimizing manner). For this, LSTDQ employs an orthogonal projection and sets

$$\hat{Q}^{\pi_m} = (\Phi(\Phi^T \Phi)^{-1} \Phi^T) \hat{H} \hat{Q}^{\pi_m}. \quad (2.11)$$

With the compactly written version of equation (2.10), $\hat{H}_{\pi_m} Q^{\pi_m} = \mathcal{R} + \gamma \mathcal{P} \Pi_{\pi_m} Q^{\pi_m}$, equation (2.11) can be rearranged to

$$w^{\pi_m} = (\Phi^T (\Phi - \gamma \mathcal{P} \Pi_{\pi_m} \Phi))^{-1} \Phi^T \mathcal{R}$$

where Π_{π_m} is a stochastic matrix of size $|S| \times |S||A|$ describing policy π_m :

$$\Pi_{\pi_m}(s, (s', a')) = \pi_m(s', a').$$

Importantly, in this equation LSTDQ approximates the model of the system on the basis of the given sample set \mathcal{F} , i.e., P is a stochastic matrix of size $|S||A| \times |S|$ that contains transition probabilities, as observed within the transition set according to

$$P((s, a), s') = \sum_{(s, a, \cdot, s') \in \mathcal{F}} 1 / \sum_{(s, a, \cdot, \cdot) \in \mathcal{F}} 1 \approx Pr(s, a, s')$$

and \mathcal{R} is a vector of size $|S||A|$ that summarizes the rewards contained in \mathcal{F} .

After having determined the state-action value function \hat{Q}^{π_m} for the current policy π_m , a greedy (improved) policy π_{m+1} can be derived as usual by letting

$$\pi_{m+1}(s) = \arg \max_{a \in A} \hat{Q}_m^\pi(s, a) = \arg \max_{a \in A} \phi(s, a)(w^{\pi_m})^T.$$

Since LSPI never stores policies explicitly, but rather implicitly by the set of basis functions and corresponding weights—and thus, in fact, by a state-action value function—the policy improvement step of LSPI merely consists of overwriting the old weight vector w with the current weight vector found by a call to LSTDQ.

Valuable insight can be obtained by comparing a single iteration of the Fitted Q Iteration algorithm with a single iteration (one policy evaluation and improvement step) of LSPI. The main difference between LSPI and value function-centered FQI algorithms is that, in a single iteration, LSPI determines an approximation of the state-action value function Q^{π_m} for the current policy and batch of experience. LSPI can do this analytically—i.e., without iterating the \hat{H}_{π_m} operator—because of the properties of its linear function approximation architecture. By contrast, FQI algorithms rely on a set of target values for the supervised fitting of a function approximator that are based on a single dynamic programming update step—i.e., on a single application of the \hat{H} operator. Consequently, if we interpret Fitted Q Iteration algorithms from a policy iteration perspective, then this class of algorithms implements a batch variant of optimistic policy iteration, whereas LSPI realizes standard (non-optimistic) policy iteration.

2.4.4 Identifying Batch Algorithms

Whereas the algorithms described here could be seen as the foundation of modern batch reinforcement learning, several other algorithms have been referred to as ‘batch’ or ‘semi-batch’ algorithms in the past. Furthermore, the borders between ‘online’, ‘offline’, ‘semi-batch’, and ‘batch’ can not be drawn distinctly; there are at least two different perspectives to look at the problem. Figure 2.4 proposes an ordering of online, semi-batch, growing batch, and batch reinforcement learning algorithms. On one side of the tree we have pure online algorithms like classic

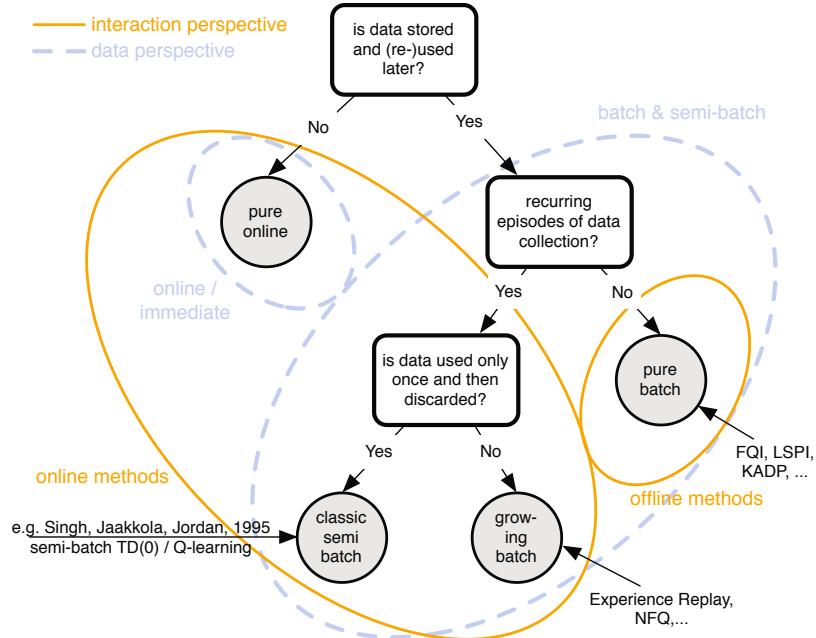


Fig. 2.4 Classification of batch vs. non-batch algorithms. With the interaction perspective and the data-usage perspective there are at least two different perspectives with which to define the category borders.

Q-learning. On the opposite side of the tree we have pure batch algorithms that work completely ‘offline’ on a fixed set of transitions. In-between these extremal positions are a number of other algorithms that, depending on the perspective, could be classified as either online or (semi-)batch algorithms. For example, the growing batch approach could be classified as an online method—it interacts with the system like an online method and incrementally improves its policy as new experience becomes available—as well as, from a data usage perspective, being seen as a batch-algorithm, since it stores all experience and uses ‘batch methods’ to learn from these observations. Although FQI—like KADP and LSPI—has been proposed by Ernst as a pure batch algorithm working on a fixed set of samples, it can easily be adapted to the growing batch setting, as, for example, shown by Kalyanakrishnan and Stone (2007). This holds true for every ‘pure’ batch approach. On the other hand, NFQ (see section 2.6.1), which has been introduced in a growing-batch setting, can also be adapted to the pure batch setting in a straight-forward manner. Another class is formed by the ‘semi-batch’ algorithms that were introduced in the 90’s (Singh et al, 1995) primarily for formal reasons. These algorithms make an aggregate update for several transitions—so it is not pure online learning with immediate updates. But what they do not do, however, is store and reuse the experience after making this update—so its not a full batch approach either.

2.5 Theory of Batch RL

The compelling feature of the batch RL approach is that it grants stable behavior for Q-learning-like update rules and a whole class of function approximators (averagers) in a broad number of systems, independent of a particular modeling or specific reward function. There are two aspects to discuss: a) stability, in the sense of guaranteed convergence to a solution and b) quality, in the sense of the distance of this solution to the true optimal value function.

Gordon (1995a,b) introduced the important notion of the ‘averager’ and proved convergence of his model-based fitted value iteration for this class of function approximation schemes by first showing their non-expansive properties (in maximum norm) and then relying on the classical contraction argument (Bertsekas and Tsitsiklis, 1996) for MDPs with discounted rewards. For non-discounted problems he identified a more restrictive class of compatible function approximators and proved convergence for the ‘self-weighted’ averagers (Gordon, 1995b, section 4). Ormoneit and Sen extended these proofs to the model-free case; their kernel-based approximators are equivalent to the ‘averagers’ introduced by Gordon (Ormoneit and Sen, 2002). Approximated values must be a weighted average of the samples, where all weights are positive and add up to one (see section 2.4.1). These requirements grant the non-expansive property in maximum norm. Ormoneit and Sen showed that their random dynamic programming operator using kernel-based approximation contracts the approximated function in maximum norm for any given set of samples and, thus, converges to a unique fixed point in the space of possible approximated functions. The proof has been carried out explicitly for MDPs with discounted rewards (Ormoneit and Sen, 2002) and average-cost problems (Ormoneit and Glynn, 2001, 2002).

Another important aspect is the quality of the solution found by the algorithms. Gordon gave an absolute upper bound on the distance of the fixed point of his fitted value iteration to the optimal value function (Gordon, 1995b). This bound depends mainly on the expressiveness of the function approximator and its ‘compatibility’ with the optimal value function to approximate. Apart from the function approximator, in model-free batch reinforcement learning the random sampling of the transitions obviously is another aspect that influences the quality of the solution. Therefore, for KADP, there is no absolute upper bound limiting the distance of the approximate solution given a particular function approximator. Ormoneit and Sen instead proved the stochastic *consistency* of their algorithm—actually, this could be seen as an even stronger statement. Continuously increasing the size of samples in the limit guarantees stochastic convergence to the optimal value function under certain assumptions (Ormoneit and Sen, 2002). These assumptions (Ormoneit and Sen, 2002, appendix A)—besides other constraints on the sampling of transitions—include smoothness constraints on the reward function (needs to be a Lipschitz continuous function of s , a and s') and the kernel. A particular kernel used throughout their experiments (Ormoneit and Glynn, 2001) that fulfills these constraints is derived from the ‘mother kernel’.

$$k_{\mathcal{F}_a,b}(s, \sigma) = \phi^+ \left(\frac{\|s - \sigma\|}{b} \right) \Bigg/ \sum_{(s_i, a_i, r_i, s'_i) \in \mathcal{F}_a} \phi^+ \left(\frac{\|s_i - \sigma\|}{b} \right)$$

with ϕ^+ being a univariate Gaussian function. The parameter b controls the ‘bandwidth’ of the kernel—that is its region of influence, or, simply, its ‘resolution’. Relying on such a kernel, the main idea of their consistency proof is to first define an ‘admissible’ reduction rate for the parameter b in dependence of the growing number of samples and then prove the stochastic convergence of the series of approximations \hat{V}^k under this reduction rate to the optimal value function. Reducing the bandwidth parameter b can be interpreted as increasing the resolution of the approximator. When reducing b , the expected deviation of the implicitly-estimated transition model from the true transition probabilities—in the limit—vanishes to zero as more and more samples become available. It is important to note that increasing the resolution of the approximator is only guaranteed to improve the approximation for smooth reward functions and will not necessarily help in approximating step functions, for example—thus, the Lipschitz constraint on the reward function.

Besides these results, which are limited to the usage of averagers within the batch RL algorithms, there are promising new theoretical analysis of Antos, Munos, and Szepesvari, that presently do not cover more general function approximators, but, however, may lead to helpful results for non-averagers in the future (Antos et al, 2008).

2.6 Batch RL in Practice

In this section, we will present several applications of batch reinforcement learning methods to real-world problems.

2.6.1 Neural Fitted Q Iteration (NFQ)

The ability to approximate functions with high accuracy and to generalize well from few training examples makes neural networks—in particular, multi-layer perceptrons (Rumelhart et al, 1986; Werbos, 1974)—an attractive candidate to represent value functions. However, in the classical online reinforcement learning setting, the current update often has unforeseeable influence on the efforts taken so far. In contrast, batch RL changes the situation dramatically: by updating the value function simultaneously at all transitions seen so far, the effect of destroying previous efforts can be overcome. This was the driving idea behind the proposal of Neural Fitted Q Iteration (NFQ, Riedmiller (2005)). As a second important consequence, the simultaneous update at all training instances makes the application of batch supervised

learning algorithms possible. In particular, within the NFQ framework, the adaptive supervised learning algorithm Rprop (Riedmiller and Braun, 1993) is used as the core of the fitting step.

```

NFQ_main() {
    input: a set  $\mathcal{F}$  of transition samples  $(s, a, r, s')$  (same, as used throughout the text)
    output: approximation  $\hat{Q}^N$  of the Q-value function
    i=0
    init_MLP()  $\rightarrow \hat{Q}^0$ ;
    Do {
        generate_pattern_set  $P = \{(input_t; target_t), t = 1, \dots, \#D\}$  where:
         $input_t = s, a,$ 
         $target_t = r + \gamma \max_{a' \in A} \hat{Q}^i(s', a')$ 
        add_artificial_patterns( $P$ )
        normalize_target_values( $P$ )
        scale_pattern_values( $P$ )
        Rprop_training( $P$ )  $\rightarrow \hat{Q}^{i+1}$ 
         $i \leftarrow i + 1$ 
    } WHILE ( $i < N$ )
}

```

Fig. 2.5 Main loop of NFQ

The implementation of the batch RL framework using neural networks is fairly straight-forward, as the algorithm in figure 2.5 shows. However, there are some additional tricks and techniques that help to overcome some of the problems that occur when approximating (Q-)value functions by multi-layer perceptrons:

- scaling input and target values is crucial for success and should always be done when using neural networks. A sensible scaling can be easily realized, since all training patterns are known at the beginning of training.
- adding artificial training patterns (also called ‘hint-to-goal’-heuristic in Riedmiller (2005)). Since the neural network generalizes from collected experiences, it can be observed that the network output tends to increase to its maximum value if no or too few goal-state experiences with zero path costs are included in the pattern set. A simple method to overcome this problem is to build additional artificial (i.e. not observed) patterns within the goal region with target value zero that literally ‘clamp’ the neural network output in that region to 0. For many problems this method is highly effective and can be easily applied. When the target region is known, which is typically the case, no extra knowledge is used in applying this method.
- the ‘Qmin-heuristic’: normalizing the ‘Q’ target values (Hafner and Riedmiller, 2011). A second method used to curtail the effect of increasing output values is to carry out a normalization step by subtracting the lowest target value from all target values. This results in a pattern set that has a target value of 0 for at



Fig. 2.6 Brainstormers MidSize league robot. The difficulty of dribbling lies in the fact, that by the rules at most one third of the ball might be covered by the robot. Not loosing the ball while turning therefore requires a sophisticated control of the robot motion.

least one training pattern. This method has the advantage in that no additional knowledge about states in the target regions need be known in advance.

- using a smooth immediate cost-function (Hafner and Riedmiller, 2011). Since multi-layer perceptrons basically realize a smooth mapping from inputs to outputs, it is reasonable to also use a smooth immediate cost function. As an example, consider the immediate cost function that gives constant positive costs outside the target region and 0 costs inside the target region. This leads to a minimum time control behavior, which is favorable in many applications. However, accordingly, the path costs have rather crispy jumps, which are kind of difficult to represent by a neural network. Replacing this immediate cost function with a smoothed version, the main characteristic of the policy induced by the crisp immediate cost function is widely preserved while the value function approximation is much smoother. For more details, see Hafner and Riedmiller (2011).

2.6.2 *NFQ in Control Applications*

Applying reinforcement learning to the control of technical processes is particularly appealing, since it promises to autonomously learn optimal or near optimal controllers even in the presence of noise or nonlinearities without knowing process behavior in advance. The introduction of batch RL has contributed to a major breakthrough in this domain, since, due to its data efficiency, it is now possible to learn complex control behavior from scratch by directly interacting with the real system. Some recent examples of NFQ in real-world applications are learning to swing-up and balance a real cart-pole system, time optimal position control of pneumatic devices, and learning to accurately steer a real car within less than half an hour of driving (Riedmiller et al, 2007).

The following briefly describes the learning of a neural dribble controller for a RoboCup MidSize League robot (for more details, see also Riedmiller et al (2009)). The autonomous robot (figure 2.6) uses a camera as its main sensor and is fitted with an omnidirectional drive. The control interval is 33 ms. Each motor command consists of three values denoting v_y^{target} (target forward speed relative to the coordinate system of the robot), v_x^{target} (target lateral speed) and $v_{\theta}^{\text{target}}$ (target rotation speed).

Dribbling means being able to keep the ball in front of the robot while turning to a given target. Because the rules of the MidSize league forbid simply grabbing the ball and only allow one-third of the ball to be covered by a dribbling device, this is quite a challenging task: the dribbling behavior must carefully control the robot, such that the ball does not get away from the robot when it changes direction.

The learning problem is modelled as a stochastic shortest path problem with both a terminal goal state and terminal failure states. Intermediate steps are punished by constant costs of 0.01.² NFQ is used as the core learning algorithm. The computation of the target value for the batch training set thus becomes:

$$\bar{q}_{s,a}^{i+1} := \begin{cases} 1.0 & , \text{ if } s' \in S^- \\ 0.01 & , \text{ if } s' \in S^+ \\ 0.01 + \min_{a' \in A} \hat{Q}^i(s', a') & , \text{ else} \end{cases} \quad (2.12)$$

where S^- denotes the states at which the ball is lost, and S^+ denotes the states at which the robot has the ball and heads towards the target. State information contains speed of the robot in relative x and y direction, rotation speed, x and y ball position relative to the robot and, finally, the heading direction relative to the given target direction. A failure state $s \in S^-$ is encountered, if the ball's relative x coordinate is larger than 50 mm or less than -50 mm, or if the relative y coordinate exceeds 100 mm. A success state is reached whenever the absolute difference between the heading angle and the target angle is less than 5 degrees.

The robot is controlled by a three-dimensional action vector denoting target translational and rotational speeds. A total of 5 different action triples are used, $U = \{(2.0, 0.0, 2.0), (2.5, 0.0, 1.5), (2.5, 1.5, 1.5), (3.0, 1.0, 1.0), (3.0, -1.0, 1.0)\}$, where each triple denotes $(v_x^{\text{target}}, v_y^{\text{target}}, v_{\theta}^{\text{target}})$.

Input to the Neural Fitted Q Iteration method is a set of transition tuples of the form (state, action, cost, successor state) where the cost has either been ‘observed’ (external source) or is calculated ‘on-the-fly’ with the help of a known cost function $c : S \times A \times S \mapsto R$ (internal source). A common procedure used to sample these transitions is to alternate between training the Q-function and then sampling new transitions episode-wise by greedily exploiting the current Q-function. However, on the real robot this means that between each data collection phase one has to wait until the new Q-function has been trained. This can be aggravating, since putting the ball back on the play-field requires human interaction. Therefore, a batch-sampling method is used, which collects data over multiple trials without relearning.

² Please note: costs = negative rewards. In this technical setting it’s more natural to minimize costs, what, in principle, is equivalent to maximizing (negative) rewards.

The value function is represented by a multi-layer perceptron with 9 input units (6 state variables and 3 action variables), 2 hidden layers of 20 neurons each and 1 output neuron. After each batch of 12 trials, 10 NFQ iterations are performed. Learning the target values can be done within 300 epochs of supervised batch learning, using the Rprop learning method with standard parameters. After the learning is finished, the new controller can be used to control the robot during a subsequent data collection phase. In our experiments, after 11 batches (= 132 trials), a very good controller was learned. The complete learning procedure took about one and a half hours, including the time used for offline updating of the neural approximation of the Q-function. Including preparation phases, the actual interaction time with the real robot was about 30 minutes.

The neural dribbling skill performed significantly better than the previously used hand-coded and hand-tuned dribbling routine, particularly in terms of space and time needed to turn to the desired target direction (see figure 2.7). The neural dribbling skill has been successfully used in the Brainstormers competition team since 2007. Using it, the Brainstormers won the RoboCup world championship 2007 in Atlanta, USA and third place at the world championship in 2008 in Suzhou, China.

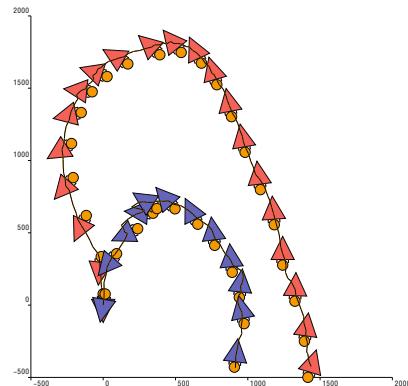


Fig. 2.7 Comparison of hand-coded (red) and neural dribbling behavior (blue) when requested to make a U-turn. The data was collected on the real robot. When the robot gets the ball, it typically has an initial speed of about 1.5 to 2 m/s in forward direction. The position of the robots are displayed every 120 ms. The U-turn performed by the neural dribbling controller is much sharper and faster.

2.6.3 Batch RL for Learning in Multi-agent Systems

While the previous two sections have pointed to the advantages of combining the data-efficiency of batch-mode RL with neural network-based function approximation schemes, this section elaborates on the benefits of batch methods for cooperative multi-agent reinforcement learning. Assuming independently learning

agents, it is obvious that those transitions experienced by one agent are strongly affected by the decisions concurrently made by other agents. This dependency of single transitions on external factors, i.e. on other agents' policies, gives rise to another argument for batch training: While a single transition tuple contains probably too little information for performing a reliable update, a rather comprehensive batch of experience may contain sufficient information to apply value function-based RL in a multi-agent context.

The framework of decentralized Markov decision processes (DEC-(PO)MDP, see chapter 15 or Bernstein et al (2002)) is frequently used to address environments populated with independent agents that have access to local state information only and thus do not know about the full, global state. The agents are independent of one another both in terms of acting as well as learning. Finding optimal solutions to these types of problems is, generally, intractable, which is why a meaningful goal is to find approximate joint policies for the ensemble of agents using model-free reinforcement learning. To this end, a local state-action value function $Q_k : S_k \times A_k$ is defined for each agent k that it successively computes, improves, and then uses to choose its local actions.

In a straightforward approach, a batch RL algorithm (in the following, the focus is put on the use of NFQ) might be run independently by each of the learning agents, thus disregarding the possible existence of other agents and making no attempts to enforce coordination across them. This approach can be interpreted as an ‘averaging projection’ with Q-values of state-action pairs collected from both, cooperating and non-cooperating agents. As a consequence, the agents' local Q_k functions underestimate the optimal joint Q -function. The following briefly describes a batch RL-based approach to sidestep that problem and points to a practical application where the resulting multi-agent learning procedure has been employed successfully (for more details, see also Gabel and Riedmiller (2008b)).

For a better estimation of the Q_k values, the inter-agent coordination mechanism introduced in Lauer and Riedmiller (2000) can be used and integrated within the framework of Fitted Q Iteration. The basic idea here is that each agent always optimistically assumes that all other agents behave optimally (though they often will not, e.g. due to exploration). Updates to the value function and policy learned are only performed when an agent is certain that a superior joint action has been executed. The performance of that coordination scheme quickly degrades in the presence of noise, which is why determinism in the DEC-MDP's state transitions must be assumed during the phase of collecting transitions. However, this assumption can be dropped when applying the policies learned.

For the multi-agent case, step 1 of FQI (cf. equation (2.6)) is modified: Each agent k collects its own transition set \mathcal{F}_k with local transitions (s_k, a_k, r_k, s'_k) . It then creates a reduced (so-called ‘optimistic’) training pattern set \mathcal{O}_k such that $|\mathcal{O}_k| \leq |P_k|$. Given a deterministic environment and the ability to reset the system to a specific initial state during data collection, the probability that agent k enters some s_k more than once is greater than zero. Hence, if a certain action $a_k \in A_k$ has been

taken multiple times in s_k , it may—because of differing local actions selected by other agents—yield very different rewards and local successor states for k . Instead of considering all tuples from \mathcal{F}_k , only those that have resulted in maximal expected rewards are used for creating \mathcal{O}_k . This means that we assume that all other agents take their best possible local action, which is—when combined with a_k —most suitable for the current global state. Accordingly, the optimistic target Q-values q_{s_k, a_k}^{i+1} for a given local state-action pair (s_k, a_k) of agent k is computed according to

$$q_{s_k, a_k}^{i+1} := \max_{\substack{(s, a, r, s') \in \mathcal{F}_k, \\ s=s_k, a=a_k}} \left(r + \gamma \max_{a' \in A_k} \hat{Q}_k^i(s', a') \right).$$

Consequently, \mathcal{O}_k realizes a partitioning of \mathcal{F}_k with respect to identical values of s_k and a_k , and q_{s_k, a_k}^{i+1} is the maximal sum of the immediate rewards and discounted expected rewards over all tuples $(s_k, a_k, \cdot, \cdot) \in \mathcal{F}_k$.

There are many applications that adhere to the problem class outlined, including production planning and resource allocation problems (Gabel and Riedmiller, 2008c). One particular class of problems that can be cast as decentralized MDPs is made up by job-shop scheduling problems (Brucker and Knust, 2005). Here, the goal is to allocate a specified number of jobs (also called tasks) to a limited number of resources (also called machines) in such a manner that some specific objective is optimized. Taking a batch reinforcement learning approach for scheduling, a learning agent is attached to each of the resources, receives local information about the set of jobs waiting for processing (features characterizing the jobs which are inputs to the neural networks used for value function approximation), and must decide which job to dispatch next. The agents interact repeatedly with the scheduling plant and they collect their own batches of transitions, which are then used to derive an improved dispatching policy by learning agent-specific state-action value functions \hat{Q}_k using the NFQ adaptation outlined above. For well-established benchmark problems, the learned dispatching policies exhibit competitive performance, and, moreover, exhibit good generalization capabilities, meaning that they can be immediately applied to modified factory layouts (Gabel and Riedmiller, 2008a).

2.6.4 Deep Fitted Q Iteration

Present reinforcement learning algorithms, in general, are still limited to solving tasks with state spaces of rather low dimensionality. For example, learning policies directly from high-dimensional visual input—e.g. raw images as captured by a camera—is still far from being possible. Usually in such a task the engineer provides a method for extracting the relevant information from the high-dimensional inputs and for encoding it in a low-dimensional feature space of a feasible size. The learning algorithm is then applied to this manually constructed feature space.

Here, the introduction of batch reinforcement learning provides new opportunities for dealing directly with high-dimensional state spaces. Consider a set of transitions $\mathcal{F} = \{(s_t, a_t, r_{t+1}, s_{t+1}) | t = 1, \dots, p\}$ where the states s are elements of a high-dimensional state space $s \in R^n$. The idea is to use an appropriate unsupervised learning method for learning a feature-extracting mapping from the data automatically. Ideally, the learned mapping $\phi : R^n \mapsto R^m$ with $m \ll n$ should encode all the ‘relevant’ information contained in a state s in the resulting feature vectors $z = \phi(s)$. The interesting thing now is, that by relying on the batch RL methods, we can combine learning of feature spaces together with learning a policy within a stable and data-efficient algorithm. Within the growing batch approach, when starting a new learning phase, we would first use the data \mathcal{F} to learn a new feature extraction mapping $\phi : R^n \mapsto R^m$ and then learn a policy in this feature space. This is done by first mapping all samples from the state space to the feature space, constructing a pattern set $\mathcal{F}_\phi \{(\phi(s), a, r, \phi(s')) | (s, a, r, s') \in \mathcal{F}\}$ in the feature space and then applying a batch algorithm such as FQI. Since all experiences are stored in the growing batch approach, it is possible to change the mapping after each episode of exploration and improve it with the newly available data. A new approximation of the value function can be calculated immediately from the mapped transitions. Actually, using the set of transitions \mathcal{F} it is possible to directly ‘translate’ an approximation \hat{Q}^ϕ that was calculated for the feature extraction ϕ to a new feature extraction ϕ' without loosing any information. We would simply apply one step of FQI with a slightly modified calculation of the target values:

$$\bar{q}_{\phi'(s), a} = r + \gamma \max_{a' \in A} \hat{Q}_{a'}^\phi(\phi(s')) .$$

When calculating the new target value for $\bar{q}_{\phi'(s), a}$ for the new feature vector $\phi'(s)$ of state s , this update uses the expected reward from the subsequent state s' as given by the old approximation \hat{Q}^ϕ using the feature vector $\phi(s')$ in the old feature space. These target values are then used to calculate a new approximation $\hat{Q}^{\phi'}$ for the new feature space.

We have already implemented this idea in a new algorithm named ‘Deep Fitted Q Iteration’ (DFQ) (Lange and Riedmiller, 2010a,b). DFQ uses a deep auto-encoder neural network (Hinton and Salakhutdinov, 2006) with up to millions of weights for unsupervised learning of low-dimensional feature spaces from high dimensional visual inputs. Training of these neural networks is embedded in a growing batch reinforcement learning algorithm derived from Fitted Q Iteration, thus enabling learning of feasible feature spaces and useful control policies at the same time (see figure 2.8). By relying on kernel-based averagers for approximating the value function in the automatically constructed feature spaces, DFQ inherits the stable learning behavior from the batch methods. Extending the theoretical results of Ormoneit and Sen, the inner loop of DFQ could be shown to converge to a unique solution for any given set of samples of any MDP with discounted rewards (Lange, 2010).

The DFQ algorithm has been successfully applied to learning visual control policies in a grid-world benchmark problem—using synthesized (Lange and Riedmiller, 2010b) as well as screen-captured images (Lange and Riedmiller, 2010a)—and to controlling a slot-car racer only on the basis of the raw image data captured by a top-mounted camera (Lange, 2010).

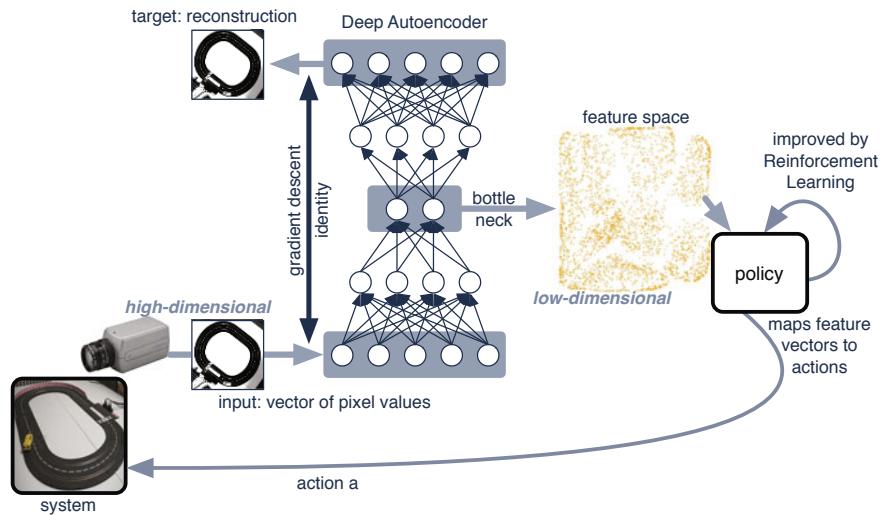


Fig. 2.8 Schematic drawing of Deep Fitted Q Iteration. Raw visual inputs from the system are fed into a deep auto-encoder neural network that learns to extract a low-dimensional encoding of the relevant information in its bottle-neck layer. The resulting feature vectors are then used to learn policies with the help of batch updates.

2.6.5 Applications/ Further References

The following table provides a quick overview of recent applications of batch RL methods.

Domain	Description	Method	Reference
Technical process control, real world	Slot car racing	NFQ	Kietzmann and Riedmiller (2009)
Technical process control, real world	Dribbling soccer robot	NFQ	Riedmiller et al (2009)
Technical process control, real world	Steering an autonomous car	NFQ	Riedmiller et al (2007)
Technical process control, simulation	Nonlinear control benchmarks	NFQ, NFQCA	Hafner and Riedmiller (2011)
Technical process control, simulation	Pole swing up and balancing	Batch RL, Gaussian Processes	Deisenroth et al (2009)
Technical process control, simulation	Mobile wheeled pendulum	NFQ, FQI (Extra Trees)	Bonarini et al (2008)
Technical process control, simulation	Semi-active suspension control	Tree based batch RL	Tognetti et al (2009)
Technical process control, simulation	Control of a power system, comparison to MPC	FQI (Extra Trees)	Ernst et al (2005b), Ernst et al (2009)
Portfolio management, simulation	Managing financial transactions	KADP	Ormoneit and Glynn (2001)
Benchmarking, simulation	Mountain car, acrobot	FQI, CMAC	Timmer and Riedmiller (2007)
Multi agent systems, simulation	Decentralized scheduling policies	NFQ	Gabel and Riedmiller (2008a)
Multi agent systems, simulation	Keepaway soccer	FQI (NN, CMAC)	Kalyanakrishnan and Stone (2007)
Medical applications	Treatment of Epilepsy	Tree based batch RL	Guez et al (2008)

2.7 Summary

This chapter has reviewed both the historical roots and early algorithms as well as contemporary approaches and applications of batch-mode reinforcement learning. Research activity in this field has grown substantially in recent years, primarily due to the central merits of the batch approach, namely, its efficient use of collected data as well as the stability of the learning process caused by the separation of the dynamic programming and value function approximation steps. Besides this, various practical implementations and applications for real-world learning tasks have contributed to the increased interest in batch RL approaches.

References

- Antos, A., Munos, R., Szepesvari, C.: Fitted Q-iteration in continuous action-space MDPs. In: Advances in Neural Information Processing Systems, vol. 20, pp. 9–16 (2008)
- Baird, L.: Residual algorithms: Reinforcement learning with function approximation. In: Proc. of the Twelfth International Conference on Machine Learning, pp. 30–37 (1995)
- Bernstein, D., Givan, D., Immerman, N., Zilberstein, S.: The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research* 27(4), 819–840 (2002)
- Bertsekas, D., Tsitsiklis, J.: Neuro-dynamic programming. Athena Scientific, Belmont (1996)
- Bonarini, A., Caccia, C., Lazaric, A., Restelli, M.: Batch reinforcement learning for controlling a mobile wheeled pendulum robot. In: IFIP AI, pp. 151–160 (2008)
- Brucker, P., Knust, S.: Complex Scheduling. Springer, Berlin (2005)
- Deisenroth, M.P., Rasmussen, C.E., Peters, J.: Gaussian Process Dynamic Programming. *Neurocomputing* 72(7-9), 1508–1524 (2009)
- Ernst, D., Geurts, P., Wehenkel, L.: Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research* 6(1), 503–556 (2005a)
- Ernst, D., Glavic, M., Geurts, P., Wehenkel, L.: Approximate Value Iteration in the Reinforcement Learning Context. Application to Electrical Power System Control. *International Journal of Emerging Electric Power Systems* 3(1) (2005b)
- Ernst, D., Glavic, M., Capitanescu, F., Wehenkel, L.: Reinforcement learning versus model predictive control: a comparison on a power system problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 39(2), 517–529 (2009)
- Gabel, T., Riedmiller, M.: Adaptive Reactive Job-Shop Scheduling with Reinforcement Learning Agents. *International Journal of Information Technology and Intelligent Computing* 24(4) (2008a)
- Gabel, T., Riedmiller, M.: Evaluation of Batch-Mode Reinforcement Learning Methods for Solving DEC-MDPs with Changing Action Sets. In: Girgin, S., Loth, M., Munos, R., Preux, P., Ryabko, D. (eds.) EWRL 2008. LNCS (LNAI), vol. 5323, pp. 82–95. Springer, Heidelberg (2008)
- Gabel, T., Riedmiller, M.: Reinforcement Learning for DEC-MDPs with Changing Action Sets and Partially Ordered Dependencies. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), IFAAMAS, Estoril, Portugal, pp. 1333–1336 (2008)
- Gordon, G.J.: Stable Function Approximation in Dynamic Programming. In: Proc. of the Twelfth International Conference on Machine Learning, pp. 261–268. Morgan Kaufmann, Tahoe City (1995a)
- Gordon, G.J.: Stable function approximation in dynamic programming. Tech. rep., CMU-CS-95-103, CMU School of Computer Science, Pittsburgh, PA (1995b)
- Gordon, G.J.: Chattering in SARSA (λ). Tech. rep. (1996)
- Guez, A., Vincent, R.D., Avoli, M., Pineau, J.: Adaptive treatment of epilepsy via batch-mode reinforcement learning. In: AAAI, pp. 1671–1678 (2008)
- Hafner, R., Riedmiller, M.: Reinforcement Learning in Feedback Control — challenges and benchmarks from technical process control. *Machine Learning* (accepted for publication, 2011), doi:10.1007/s10994-011-5235-x
- Hinton, G., Salakhutdinov, R.: Reducing the Dimensionality of Data with Neural Networks. *Science* 313(5786), 504–507 (2006)

- Kalyanakrishnan, S., Stone, P.: Batch reinforcement learning in a complex domain. In: The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 650–657. ACM, New York (2007)
- Kietzmann, T., Riedmiller, M.: The Neuro Slot Car Racer: Reinforcement Learning in a Real World Setting. In: Proceedings of the Int. Conference on Machine Learning Applications (ICMLA 2009). Springer, Miami (2009)
- Lagoudakis, M., Parr, R.: Model-Free Least-Squares Policy Iteration. In: Advances in Neural Information Processing Systems, vol. 14, pp. 1547–1554 (2001)
- Lagoudakis, M., Parr, R.: Least-Squares Policy Iteration. *Journal of Machine Learning Research* 4, 1107–1149 (2003)
- Lange, S.: Tiefes Reinforcement Lernen auf Basis visueller Wahrnehmungen. Dissertation, Universität Osnabrück (2010)
- Lange, S., Riedmiller, M.: Deep auto-encoder neural networks in reinforcement learning. In: International Joint Conference on Neural Networks (IJCNN 2010), Barcelona, Spain (2010a)
- Lange, S., Riedmiller, M.: Deep learning of visual control policies. In: European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2010), Brugge, Belgium (2010b)
- Lauer, M., Riedmiller, M.: An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems. In: Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), pp. 535–542. Morgan Kaufmann, Stanford (2000)
- Lin, L.: Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning* 8(3), 293–321 (1992)
- Ormoneit, D., Glynn, P.: Kernel-based reinforcement learning in average-cost problems: An application to optimal portfolio choice. In: Advances in Neural Information Processing Systems, vol. 13, pp. 1068–1074 (2001)
- Ormoneit, D., Glynn, P.: Kernel-based reinforcement learning in average-cost problems. *IEEE Transactions on Automatic Control* 47(10), 1624–1636 (2002)
- Ormoneit, D., Sen, Š.: Kernel-based reinforcement learning. *Machine Learning* 49(2), 161–178 (2002)
- Riedmiller, M.: Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 317–328. Springer, Heidelberg (2005)
- Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: Rusipini, H. (ed.) Proceedings of the IEEE International Conference on Neural Networks (ICNN), San Francisco, pp. 586–591 (1993)
- Riedmiller, M., Montemerlo, M., Dahlkamp, H.: Learning to Drive in 20 Minutes. In: Proceedings of the FBIT 2007 Conference. Springer, Jeju (2007)
- Riedmiller, M., Hafner, R., Lange, S., Lauer, M.: Learning to dribble on a real robot by success and failure. In: Proc. of the IEEE International Conference on Robotics and Automation, pp. 2207–2208 (2008)
- Riedmiller, M., Gabel, T., Hafner, R., Lange, S.: Reinforcement Learning for Robot Soccer. *Autonomous Robots* 27(1), 55–74 (2009)
- Rumelhart, D., Hinton, G., Williams, R.: Learning representations by back-propagating errors. *Nature* 323(6088), 533–536 (1986)
- Schoknecht, R., Merke, A.: Convergent combinations of reinforcement learning with linear function approximation. In: Advances in Neural Information Processing Systems, vol. 15, pp. 1611–1618 (2003)

- Singh, S., Jaakkola, T., Jordan, M.: Reinforcement learning with soft state aggregation. In: Advances in Neural Information Processing Systems, vol. 7, pp. 361–368 (1995)
- Sutton, R., Barto, A.: Reinforcement Learning. An Introduction. MIT Press/A Bradford Book, Cambridge, USA (1998)
- Timmer, S., Riedmiller, M.: Fitted Q Iteration with CMACs. In: Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007), Honolulu, USA (2007)
- Tognetti, S., Savaresi, S., Spelta, C., Restelli, M.: Batch reinforcement learning for semi-active suspension control, pp. 582–587 (2009)
- Werbos, P.: Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University (1974)

Chapter 3

Least-Squares Methods for Policy Iteration

Lucian Buşoniu*, Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos,
Robert Babuška, and Bart De Schutter

Abstract. Approximate reinforcement learning deals with the essential problem of applying reinforcement learning in large and continuous state-action spaces, by using function approximators to represent the solution. This chapter reviews least-squares methods for policy iteration, an important class of algorithms for approximate reinforcement learning. We discuss three techniques for solving the core, policy evaluation component of policy iteration, called: least-squares temporal difference, least-squares policy evaluation, and Bellman residual minimization. We introduce these techniques starting from their general mathematical principles and detailing them down to fully specified algorithms. We pay attention to online variants of policy iteration, and provide a numerical example highlighting the behavior of representative offline and online methods. For the policy evaluation component as well as for the overall resulting approximate policy iteration, we provide guarantees on the performance obtained asymptotically, as the number of samples processed and iterations executed grows to infinity. We also provide finite-sample results, which apply when a finite number of samples and iterations are considered. Finally, we outline several extensions and improvements to the techniques and methods reviewed.

Lucian Buşoniu · Alessandro Lazaric · Mohammad Ghavamzadeh · Rémi Munos
Team SequeL, INRIA Lille-Nord Europe, France
e-mail: {ion-lucian.busoniu, alessandro.lazaric}@inria.fr,
{mohammad.ghavamzadeh, remi.munos}@inria.fr

Robert Babuška · Bart De Schutter
Delft Center for Systems and Control, Delft University of Technology, The Netherlands
e-mail: {r.babuska, b.deschutter}@tudelft.nl

* This work was performed in part while Lucian Buşoniu was with the Delft Center for Systems and Control.

3.1 Introduction

Policy iteration is a core procedure for solving reinforcement learning problems, which evaluates policies by estimating their value functions, and then uses these value functions to find new, improved policies. In Chapter 2, the classical policy iteration was introduced, which employs tabular, exact representations of the value functions and policies. However, most problems of practical interest have state and action spaces with a very large or even infinite number of elements, which precludes tabular representations and exact policy iteration. Instead, *approximate policy iteration* must be used. In particular, approximate policy evaluation – constructing approximate value functions for the policies considered – is the central, most challenging component of approximate policy iteration. While representing the policy can also be challenging, an explicit representation is often avoided, by computing policy actions on-demand from the approximate value function.

Some of the most powerful state-of-the-art algorithms for approximate policy evaluation represent the value function using a linear parameterization, and obtain a linear system of equations in the parameters, by exploiting the linearity of the Bellman equation satisfied by the value function. Then, in order to obtain parameters approximating the value function, this system is solved in a least-squares sample-based sense, either in one shot or iteratively.

Since highly efficient numerical methods are available to solve such systems, least-squares methods for policy evaluation are computationally efficient. Additionally taking advantage of the generally fast convergence of policy iteration methods, an overall fast policy iteration algorithm is obtained. More importantly, least-squares methods are sample-efficient, i.e., they approach their solution quickly as the number of samples they consider increases. This is a crucial property in reinforcement learning for real-life systems, as data obtained from such systems are very expensive (in terms of time taken for designing and running data collection experiments, of system wear-and-tear, and possibly even of economic costs).

In this chapter, we review least-squares methods for policy iteration: the class of approximate policy iteration methods that employ least-squares techniques at the policy evaluation step. The review is organized as follows. Section 3.2 provides a quick recapitulation of classical policy iteration, and also serves to further clarify the technical focus of the chapter. Section 3.3 forms the chapter’s core, thoroughly introducing the family of least-squares methods for policy evaluation. Section 3.4 describes an application of a particular algorithm, called simply least-squares policy iteration, to online learning. Section 3.5 illustrates the behavior of offline and online least-squares policy iteration in an example. Section 3.6 reviews available theoretical guarantees about least-squares policy evaluation and the resulting, approximate policy iteration. Finally, Section 3.7 outlines several important extensions and improvements to least-squares methods, and mentions other reviews of reinforcement learning that provide different perspectives on least-squares methods.

3.2 Preliminaries: Classical Policy Iteration

In this section, we revisit the classical policy iteration algorithm and some relevant theoretical results from Chapter 2, adapting their presentation for the purposes of the present chapter.

Recall first some notation. A Markov decision process with states $s \in S$ and actions $a \in A$ is given, governed by the stochastic dynamics $s' \sim T(s, a, \cdot)$ and with the immediate performance described by the rewards $r = R(s, a, s')$, where T is the transition function and R the reward function. The goal is to find an optimal policy $\pi^* : S \rightarrow A$ that maximizes the value function $V^\pi(s)$ or $Q^\pi(s, a)$. For clarity, in this chapter we refer to state value functions V as “V-functions”, thus achieving consistency with the name “Q-functions” traditionally used for state-action value functions Q . We use the name “value function” to refer collectively to V-functions and Q-functions.

Policy iteration works by iteratively evaluating and improving policies. At the *policy evaluation* step, the V-function or Q-function of the current policy is found. Then, at the *policy improvement* step, a new, better policy is computed based on this V-function or Q-function. The procedure continues afterward with the next iteration. When the state-action space is finite and exact representations of the value function and policy are used, policy improvement obtains a strictly better policy than the previous one, unless the policy is already optimal. Since additionally the number of possible policies is finite, policy iteration is guaranteed to find the optimal policy in a finite number of iterations. Algorithm 6 shows the classical, offline policy iteration for the case when Q-functions are used.

```

1: input initial policy  $\pi_0$ 
2:  $k \leftarrow 0$ 
3: repeat
4:   find  $Q^{\pi_k}$ 
5:    $\pi_{k+1}(s) \leftarrow \arg \max_{a \in A} Q^{\pi_k}(s, a)$  {policy evaluation}
6:    $k \leftarrow k + 1$  {policy improvement}
7: until  $\pi_k = \pi_{k-1}$ 
8: output  $\pi^* = \pi_k, Q^* = Q^{\pi_k}$ 
```

Algorithm 6. Policy iteration with Q-functions.

At the policy evaluation step, the Q-function Q^π of policy π can be found using the fact that it satisfies the Bellman equation:

$$Q^\pi = B_Q^\pi(Q^\pi) \quad (3.1)$$

where the Bellman mapping (also called backup operator) B_Q^π is defined for any Q-function as follows:

$$[B_Q^\pi(Q)](s, a) = \mathbb{E}_{s' \sim T(s, a, \cdot)} \{R(s, a, s') + \gamma Q(s', \pi(s'))\} \quad (3.2)$$

Similarly, the V-function V^π of policy π satisfies the Bellman equation:

$$V^\pi = B_V^\pi(V^\pi) \quad (3.3)$$

where the Bellman mapping B_V^π is defined by:

$$[B_V^\pi(V)](s) = \mathbb{E}_{s' \sim T(s, \pi(s), \cdot)} \{R(s, \pi(s), s') + \gamma V(s')\} \quad (3.4)$$

Note that both the Q-function and the V-function are bounded in absolute value by $V_{\max} = \frac{\|R\|_\infty}{1-\gamma}$, where $\|R\|_\infty$ is the maximum absolute reward. A number of algorithms are available for computing Q^π or V^π , based, e.g., on directly solving the linear system of equations (3.1) or (3.3) to obtain the Q-values (or V-values), on turning the Bellman equation into an iterative assignment, or on temporal-difference, model-free estimation – see Chapter 2 for details.

Once Q^π or V^π is available, policy improvement can be performed. In this context, an important difference between using Q-functions and V-functions arises. When Q-functions are used, policy improvement involves only a maximization over the action space:

$$\pi_{k+1}(s) \leftarrow \arg \max_{a \in A} Q^{\pi_k}(s, a) \quad (3.5)$$

(see again Algorithm 6), whereas policy improvement with V-functions additionally requires a model, in the form of T and R , to investigate the transitions generated by each action:

$$\pi_{k+1}(s) \leftarrow \arg \max_{a \in A} \mathbb{E}_{s' \sim T(s, a, \cdot)} \{R(s, a, s') + \gamma V(s')\} \quad (3.6)$$

This is an important point in favor of using Q-functions in practice, especially for model-free algorithms.

Classical policy iteration requires exact representations of the value functions, which can generally only be achieved by storing distinct values for every state-action pair (in the case of Q-functions) or for every state (V-functions). When some of the variables have a very large or infinite number of possible values, e.g., when they are continuous, exact representations become impossible and value functions must be *approximated*. This chapter focuses on *approximate policy iteration*, and more specifically, on algorithms for approximate policy iteration that use a class of *least-squares* methods for policy evaluation.

While representing policies in large state spaces is also challenging, an explicit representation of the policy can fortunately often be avoided. Instead, improved policies can be computed on-demand, by applying (3.5) or (3.6) at every state where an action is needed. This means that an implicit policy representation – via the value function – is used. In the sequel, we focus on this setting, additionally requiring that *exact* policy improvements are performed, i.e., that the action returned is always an exact maximizer in (3.5) or (3.6). This requirement can be satisfied, e.g., when the

action space is discrete and contains not too large a number of actions. In this case, policy improvement can be performed by computing the value function for all the discrete actions, and finding the maximum among these values using enumeration.²

In what follows, wherever possible, we will introduce the results and algorithms in terms of Q-functions, motivated by the practical advantages they provide in the context of policy improvement. However, most of these results and algorithms directly extend to the case of V-functions.

3.3 Least-Squares Methods for Approximate Policy Evaluation

In this section we consider the problem of policy evaluation, and we introduce least-squares methods to solve this problem. First, in Section 3.3.1, we discuss the high-level principles behind these methods. Then, we progressively move towards the methods' practical implementation. In particular, in Section 3.3.2 we derive idealized, model-based versions of the algorithms for the case of linearly parameterized approximation, and in Section 3.3.3 we outline their realistic, model-free implementations. To avoid detracting from the main line, we postpone the discussion of most literature references until Section 3.3.4.

3.3.1 Main Principles and Taxonomy

In problems with large or continuous state-action spaces, value functions cannot be represented exactly, but must be approximated. Since the solution of the Bellman equation (3.1) will typically not be representable by the chosen approximator, the Bellman equation must be solved approximately instead. Two main classes of least-squares methods for policy evaluation can be distinguished by the approach they take to approximately solve the Bellman equation, as shown in Figure 3.1: projected policy evaluation, and Bellman residual minimization.

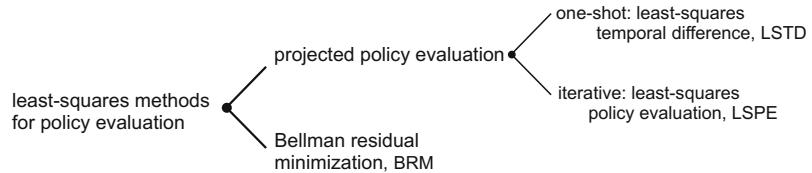


Fig. 3.1 Taxonomy of the methods covered

² In general, when the action space is large or continuous, the maximization problems (3.5) or (3.6) may be too involved to solve exactly, in which case only *approximate* policy improvements can be made.

Methods for *projected policy evaluation* look for a \hat{Q} that is approximately equal to the projection of its backed-up version $B_Q^\pi(\hat{Q})$ on the space of representable Q-functions:

$$\hat{Q} \approx \Pi(B_Q^\pi(\hat{Q})) \quad (3.7)$$

where $\Pi : \mathcal{Q} \rightarrow \widehat{\mathcal{Q}}$ denotes projection from the space \mathcal{Q} of all Q-functions onto the space $\widehat{\mathcal{Q}}$ of representable Q-functions. Solving this equation is the same as minimizing the distance between \hat{Q} and $\Pi(B_Q^\pi(\hat{Q}))$:³

$$\min_{\hat{Q} \in \widehat{\mathcal{Q}}} \|\hat{Q} - \Pi(B_Q^\pi(\hat{Q}))\| \quad (3.8)$$

where $\|\cdot\|$ denotes a generic norm/measure of magnitude. An example of such a norm will be given in Section 3.3.2. The relation (3.7) is called the *projected Bellman equation*, hence the name projected policy evaluation. As will be seen in Section 3.3.2, in the case of linear approximation (3.7) can in fact be solved exactly, i.e. with equality (whereas in general, there may not exist a function \hat{Q} that when passed through B_Q^π and Π leads exactly back to itself).

Two subclasses of methods employing the principle of projected policy evaluation can be further identified: *one-shot* methods that aim directly for a solution to the projected Bellman equation; and methods that find a solution in an *iterative* fashion. Such methods are called, respectively, least-squares temporal difference (LSTD) (Bradtko and Barto, 1996; Boyan, 2002) and least-squares policy evaluation (LSPE) (Bertsekas and Ioffe, 1996) in the literature. Note that the LSPE iterations are internal to the policy evaluation step, contained within the larger iterations of the overall policy iteration algorithm.

Methods for *Bellman residual minimization* (BRM) do not employ projection, but try to directly solve the Bellman equation in an approximate sense:

$$\hat{Q} \approx B_Q^\pi(\hat{Q})$$

by performing the minimization:

$$\min_{\hat{Q} \in \widehat{\mathcal{Q}}} \|\hat{Q} - B_Q^\pi(\hat{Q})\| \quad (3.9)$$

The difference $\hat{Q} - B_Q^\pi(\hat{Q})$ is called Bellman residual, hence the name BRM.

In the sequel, we will append the suffix “Q” to the acronym of the methods that find Q-functions (e.g., LSTD-Q, LSPE-Q, BRM-Q), and the suffix “V” in the case of V-functions. We will use the unqualified acronym to refer to both types of methods collectively, when the distinction between them is not important.

³ In this equation as well as the others, a multistep version of the Bellman mapping B_Q^π (or B_V^π) can also be used, parameterized by a scalar $\lambda \in [0, 1]$. In this chapter, we mainly consider the single-step case, for $\lambda = 0$, but we briefly discuss the multistep case in Section 3.7.

3.3.2 The Linear Case and Matrix Form of the Equations

Until now, the approximator of the value function, as well as the norms in the minimizations (3.8) and (3.9) have been left unspecified. The most common choices are, respectively, approximators linear in the parameters and (squared, weighted) Euclidean norms, as defined below. With these choices, the minimization problems solved by projected and BRM policy evaluation can be written in terms of matrices and vectors, and have closed-form solutions that eventually lead to efficient algorithms. The name “least-squares” for this class of methods comes from minimizing the squared Euclidean norm.

To formally introduce these choices and the solutions they lead to, the state and action spaces will be assumed finite, $S = \{s_1, \dots, s_N\}$, $A = \{a_1, \dots, a_M\}$. Nevertheless, the final, practical algorithms obtained in Section 3.3.3 below can also be applied to infinite and continuous state-action spaces.

A linearly parameterized representation of the Q-function has the following form:

$$\hat{Q}(s, a) = \sum_{l=1}^d \varphi_l(s, a) \theta_l = \phi^\top(s, a) \theta \quad (3.10)$$

where $\theta \in \mathbb{R}^d$ is the parameter vector and $\phi(s, a) = [\varphi_1(s, a), \dots, \varphi_d(s, a)]^\top$ is a vector of basis functions (BFs), also known as features (Bertsekas and Tsitsiklis, 1996).⁴ Thus, the space of representable Q-functions is the span of the BFs, $\hat{\mathcal{Q}} = \{\phi^\top(\cdot, \cdot) \theta \mid \theta \in \mathbb{R}^d\}$.

Given a weight function $\rho : S \times A \rightarrow [0, 1]$, the (squared) weighted Euclidean norm of a Q-function is defined by:

$$\|Q\|_\rho^2 = \sum_{\substack{i=1, \dots, N \\ j=1, \dots, M}} \rho(s_i, a_j) |Q(s_i, a_j)|^2$$

Note the norm itself, $\|Q\|_\rho$, is the square root of this expression; we will mostly use the squared variant in the sequel.

The corresponding weighted least-squares projection operator, used in projected policy evaluation, is:

$$\Pi^\rho(Q) = \arg \min_{\hat{Q}} \|\hat{Q} - Q\|_\rho^2$$

The weight function is interpreted as a probability distribution over the state-action space, so it must sum up to 1. The distribution given by ρ will be used to generate samples used by the model-free policy evaluation algorithms of Section 3.3.3 below.

Matrix Form of the Bellman Mapping

To specialize projection-based and BRM methods to the linear approximation, Euclidean-norm case, we will write the Bellman mapping (3.2) in a matrix form, and then in terms of the parameter vector.

⁴ Throughout the chapter, column vectors are employed.

Because the state-action space is discrete, the Bellman mapping can be written as a sum:

$$[B_Q^\pi(Q)](s_i, a_j) = \sum_{i'=1}^N T(s_i, a_j, s_{i'}) [R(s_i, a_j, s_{i'}) + \gamma Q(s_{i'}, \pi(s_{i'}))] \quad (3.11)$$

$$= \sum_{i'=1}^N T(s_i, a_j, s_{i'}) R(s_i, a_j, s_{i'}) + \gamma \sum_{i'=1}^N T(s_i, a_j, s_{i'}) Q(s_{i'}, \pi(s_{i'})) \quad (3.12)$$

for any i, j . The two-sum expression leads us to a matrix form of this mapping:

$$B_Q^\pi(Q) = R + \gamma T^\pi Q \quad (3.13)$$

where $B_Q^\pi : \mathbb{R}^{NM} \rightarrow \mathbb{R}^{NM}$. Denoting by $[i, j] = i + (j - 1)N$ the scalar index corresponding to i and j ,⁵ the vectors and matrices on the right hand side of (3.13) are defined as follows:⁶

- $Q \in \mathbb{R}^{NM}$ is a vector representation of the Q-function Q , with $Q_{[i, j]} = Q(s_i, a_j)$.
- $R \in \mathbb{R}^{NM}$ is a vector representation of the *expectation* of the reward function R , where the element $R_{[i, j]}$ is the expected reward after taking action a_j in state s_i , i.e., $R_{[i, j]} = \sum_{i'=1}^N T(s_i, a_j, s_{i'}) R(s_i, a_j, s_{i'})$. So, for the first sum in (3.12), the transition function T has been integrated into R (unlike for the second sum).
- $T^\pi \in \mathbb{R}^{NM \times NM}$ is a matrix representation of the transition function combined with the policy, with $T_{[i, j], [i', j']} = T(s_i, a_j, s_{i'})$ if $\pi(s_{i'}) = a_{j'}$, and 0 otherwise. A useful way to think about T^π is as containing transition probabilities between *state-action pairs*, rather than just states. In this interpretation, $T_{[i, j], [i', j']}^\pi$ is the probability of moving from an arbitrary state-action pair (s_i, a_j) to a next state-action pair $(s_{i'}, a_{j'})$ that follows the current policy. Thus, if $a_{j'} \neq \pi(s_{i'})$, then the probability is zero, which is what the definition says. This interpretation also indicates that stochastic policies can be represented with a simple modification: in that case, $T_{[i, j], [i', j']}^\pi = T(s_i, a_j, s_{i'}) \cdot \pi(s_{i'}, a_{j'})$, where $\pi(s, a)$ is the probability of taking a in s .

The next step is to rewrite the Bellman mapping in terms of the parameter vector, by replacing the generic Q-vector in (3.13) by an approximate, parameterized Q-vector. This is useful because in all the methods, the Bellman mapping is always applied to approximate Q-functions. Using the following matrix representation of the BFs:

$$\Phi_{[i, j], l} = \varphi_l(s_i, a_j), \quad \Phi \in \mathbb{R}^{NM \times d} \quad (3.14)$$

⁵ If the d elements of the BF vector were arranged into an $N \times M$ matrix, by first filling in the first column with the first N elements, then the second column with the subsequent N elements, etc., then the element at index $[i, j]$ of the vector would be placed at row i and column j of the matrix.

⁶ Boldface notation is used for vector or matrix representations of functions and mappings. Ordinary vectors and matrices are displayed in normal font.

an approximate Q-vector is written $\hat{Q} = \Phi\theta$. Plugging this into (3.13), we get:

$$B_Q^\pi(\Phi\theta) = R + \gamma T^\pi \Phi\theta \quad (3.15)$$

We are now ready to describe projected policy evaluation and BRM in the linear case. Note that the matrices and vectors in (3.15) are too large to be used directly in an implementation; however, we will see that starting from these large matrices and vectors, both the projected policy evaluation and the BRM solutions can be written in terms of smaller matrices and vectors, which can be stored and manipulated in practical algorithms.

Projected Policy Evaluation

Under appropriate conditions on the BFs and the weights ρ (see Section 3.6.1 for a discussion), the projected Bellman equation can be exactly solved in the linear case:

$$\hat{Q} = \Pi^\rho(B_Q^\pi(\hat{Q}))$$

so that a minimum of 0 is attained in the problem $\min_{\hat{Q} \in \hat{\mathcal{Q}}} \|\hat{Q} - \Pi(B_Q^\pi(\hat{Q}))\|$. In matrix form, the projected Bellman equation is:

$$\begin{aligned} \Phi\theta &= \Pi^\rho B_Q^\pi(\Phi\theta) \\ &= \Pi^\rho(R + \gamma T^\pi \Phi\theta) \end{aligned} \quad (3.16)$$

where Π^ρ is a closed-form, matrix representation of the weighted least-squares projection Π^ρ :

$$\Pi^\rho = \Phi(\Phi^\top \rho \Phi)^{-1} \Phi^\top \rho \quad (3.17)$$

The weight matrix ρ collects the weights of each state-action pair on its main diagonal:

$$\rho_{[i,j],[i,j]} = \rho(s_i, a_j), \quad \rho \in \mathbb{R}^{NM \times NM} \quad (3.18)$$

After substituting (3.17) into (3.16), a left-multiplication by $\Phi^\top \rho$, and a rearrangement of terms, we obtain:

$$\Phi^\top \rho \Phi \theta = \gamma \Phi^\top \rho T^\pi \Phi \theta + \Phi^\top \rho R$$

or in condensed form:

$$A\theta = \gamma B\theta + b \quad (3.19)$$

with the notations $A = \Phi^\top \rho \Phi$, $B = \Phi^\top \rho T^\pi \Phi$, and $b = \Phi^\top \rho R$. The matrices A and B are in $\mathbb{R}^{d \times d}$, while b is a vector in \mathbb{R}^d . This is a crucial expression, highlighting that the projected Bellman equation can be represented and solved using only small matrices and vectors (of size $d \times d$ and d), instead of the large matrices and vectors (of sizes up to $NM \times NM$) that originally appeared in the formulas.

Next, two idealized algorithms for projected policy evaluation are introduced, which assume knowledge of A , B , and b . The next section will show how this assumption can be removed.

The idealized LSTD-Q belongs to the first (“one-shot”) subclass of methods for projected policy evaluation in Figure 3.1. It simply solves the system (3.19) to arrive at the parameter vector θ . This parameter vector provides an approximate Q-function $\hat{Q}^\pi(s, a) = \phi^\top(s, a)\theta$ of the considered policy π . Note that because θ appears on both sides of (3.19), this equation can be simplified to:

$$(A - \gamma B)\theta = b$$

The idealized LSPE-Q is an iterative algorithm, thus belonging to the second subclass of projected policy evaluation methods in Figure 3.1. It also relies on (3.19), but updates the parameter vector incrementally:

$$\begin{aligned} \theta_{\tau+1} &= \theta_\tau + \alpha(\theta'_{\tau+1} - \theta_\tau) \\ \text{where } A\theta'_{\tau+1} &= \gamma B\theta_\tau + b \end{aligned} \tag{3.20}$$

starting from some initial value θ_0 . In this update, α is a positive step size parameter.

Bellman Residual Minimization

Consider the problem solved by BRM-Q (3.9), specialized to the weighted Euclidean norm used in this section:

$$\min_{\hat{Q} \in \hat{\mathcal{Q}}} \left\| \hat{Q} - B_Q^\pi(\hat{Q}) \right\|_\rho^2 \tag{3.21}$$

Using the matrix expression (3.15) for B^π , this minimization problem can be rewritten in terms of the parameter vector as follows:

$$\begin{aligned} \min_{\theta \in \mathbb{R}^d} & \| \Phi\theta - R - \gamma T^\pi \Phi\theta \|_\rho^2 \\ &= \min_{\theta \in \mathbb{R}^d} \| (\Phi - \gamma T^\pi \Phi)\theta - R \|_\rho^2 \\ &= \min_{\theta \in \mathbb{R}^d} \| C\theta - R \|_\rho^2 \end{aligned}$$

where $C = \Phi - \gamma T^\pi \Phi$. By linear-algebra arguments, the minimizer of this problem can be found by solving the equation:

$$C^\top \rho C \theta = C^\top \rho R \tag{3.22}$$

The idealized BRM-Q algorithm consists of solving this equation to arrive at θ , and thus to an approximate Q-function of the policy considered.

It is useful to note that BRM is closely related to projected policy evaluation, as it can be interpreted as solving a similar projected equation (Bertsekas, 2011a). Specifically, finding a minimizer in (3.21) is equivalent to solving:

$$\Phi\theta = \Pi^\rho(\bar{B}_Q^\pi(\Phi\theta))$$

which is similar to the projected Bellman equation (3.16), except that the modified mapping \bar{B}_Q^π is used:

$$\bar{B}_Q^\pi(\Phi\theta) = B_Q^\pi(\Phi\theta) + \gamma(T^\pi)^\top [\Phi\theta - B_Q^\pi(\Phi\theta)]$$

3.3.3 Model-Free Implementations

To obtain the matrices and vectors appearing in their equations, the idealized algorithms given above would require the transition and reward functions of the Markov decision process, which are unknown in a reinforcement learning context. Moreover, the algorithms would need to iterate over all the state-action pairs, which is impossible in the large state-action spaces they are intended for.

This means that, in practical reinforcement learning, *sample-based* versions of these algorithms should be employed. Fortunately, thanks to the special structure of the matrices and vectors involved, they can be estimated from samples. In this section, we derive practically implementable LSTD-Q, LSPE-Q, and BRM-Q algorithms.

Projected Policy Evaluation

Consider a set of n transition samples of the form (s_i, a_i, s'_i, r_i) , $i = 1, \dots, n$, where s'_i is drawn from the distribution $T(s_i, a_i, \cdot)$ and $r_i = R(s_i, a_i, s'_i)$. The state-action pairs (s_i, a_i) must be drawn from the distribution given by the weight function ρ . For example, when a generative model is available, samples can be drawn independently from ρ and the model can be used to find corresponding next states and rewards. From an opposite perspective, it can also be said that the distribution of the state-action pairs implies the weights used in the projected Bellman equation. For example, collecting samples along a set of trajectories of the system, or along a single trajectory, implicitly leads to a distribution (weights) ρ .

Estimates of the matrices A and B and of the vector b appearing in (3.19) and (3.20) can be constructed from the samples as follows:

$$\begin{aligned}\hat{A}_i &= \hat{A}_{i-1} + \phi(s_i, a_i)\phi^\top(s_i, a_i) \\ \hat{B}_i &= \hat{B}_{i-1} + \phi(s_i, a_i)\phi^\top(s'_i, \pi(s'_i)) \\ \hat{b}_i &= \hat{b}_{i-1} + \phi(s_i, a_i)r_i\end{aligned}\tag{3.23}$$

starting from zero initial values ($\hat{A}_0 = 0, \hat{B}_0 = 0, \hat{b}_0 = 0$).

LSTD-Q processes the n samples using (3.23) and then solves the equation:

$$\frac{1}{n} \hat{A}_n \theta = \gamma \frac{1}{n} \hat{B}_n \theta + \frac{1}{n} \hat{b}_n \quad (3.24)$$

or, equivalently:

$$\frac{1}{n} (\hat{A}_n - \gamma \hat{B}_n) \theta = \frac{1}{n} \hat{b}_n$$

to find a parameter vector θ . Note that, because this equation is an approximation of (3.19), the parameter vector θ is only an approximation to the solution of (3.19) (however, for notation simplicity we denote it in the same way). The divisions by n , while not mathematically necessary, increase the numerical stability of the algorithm, by preventing the coefficients from growing too large as more samples are processed. The composite matrix $(\hat{A} - \gamma \hat{B})$ can also be updated as a single entity $(\widehat{A - \gamma B})$, thereby eliminating the need of storing in memory two potentially large matrices \hat{A} and \hat{B} .

Algorithm 7 summarizes this more memory-efficient variant of LSTD-Q. Note that the update of $(\widehat{A - \gamma B})$ simply accumulates the updates of \hat{A} and \hat{B} in (3.23), where the term from \hat{B} is properly multiplied by $-\gamma$.

```

1: input policy to evaluate  $\pi$ , BFs  $\phi$ , samples  $(s_i, a_i, s'_i, r_i), i = 1, \dots, n$ 
2:  $(\widehat{A - \gamma B})_0 \leftarrow 0, \hat{b}_0 \leftarrow 0$ 
3: for  $i = 1, \dots, n$  do
4:    $(\widehat{A - \gamma B})_i \leftarrow (\widehat{A - \gamma B})_{i-1} + \phi(s_i, a_i) \phi^\top(s_i, a_i) - \gamma \phi(s_i, a_i) \phi^\top(s'_i, \pi(s'_i))$ 
5:    $\hat{b}_i \leftarrow \hat{b}_{i-1} + \phi(s_i, a_i) r_i$ 
6:   solve  $\frac{1}{n} (\widehat{A - \gamma B})_n \theta = \frac{1}{n} \hat{b}_n$ 
7: output Q-function  $\hat{Q}^\pi(s, a) = \phi^\top(s, a) \theta$ 
```

Algorithm 7. LSTD-Q

Combining LSTD-Q policy evaluation with exact policy improvements leads to what is perhaps the most widely used policy iteration algorithm from the least-squares class, called simply *least-squares policy iteration* (LSPI).

LSPE-Q uses the same estimates \hat{A} , \hat{B} , and \hat{b} as LSTD-Q, but updates the parameter vector iteratively. In its basic variant, LSPE-Q starts with an arbitrary initial parameter vector θ_0 and updates using:

$$\begin{aligned} \theta_i &= \theta_{i-1} + \alpha(\theta'_i - \theta_{i-1}) \\ \text{where } \frac{1}{i} \hat{A}_i \theta'_i &= \gamma \frac{1}{i} \hat{B}_i \theta_{i-1} + \frac{1}{i} \hat{b}_i \end{aligned} \quad (3.25)$$

This update is an approximate, sample-based version of the idealized version (3.20). Similarly to LSTD-Q, the divisions by i increase the numerical stability of the

updates. The estimate \hat{A} may not be invertible at the start of the learning process, when only a few samples have been processed. A practical solution to this issue is to initialize \hat{A} to a small multiple of the identity matrix.

A more flexible algorithm than (3.25) can be obtained by (i) processing more than one sample in-between consecutive updates of the parameter vector, as well as by (ii) performing more than one parameter update after processing each (batch of) samples, while holding the coefficient estimates constant. The former modification may increase the stability of the algorithm, particularly in the early stages of learning, while the latter may accelerate its convergence, particularly in later stages as the estimates \hat{A} , \hat{B} , and \hat{b} become more precise.

Algorithm 8 shows this more flexible variant of LSPE-Q, which (i) processes batches of \bar{n} samples in-between parameter update episodes (\bar{n} should preferably be a divisor of n). At each such episode, (ii) the parameter vector is updated N_{upd} times. Notice that unlike in (3.25), the parameter update index τ is different from the sample index i , since the two indices are no longer advancing synchronously.

```

1: input policy to evaluate  $\pi$ , BFs  $\phi$ , samples  $(s_i, a_i, s'_i, r_i)$ ,  $i = 1, \dots, n$ 
   step size  $\alpha$ , small constant  $\delta_A > 0$ 
   batch length  $\bar{n}$ , number of consecutive parameter updates  $N_{\text{upd}}$ 
2:  $\hat{A}_0 \leftarrow \delta_A I$ ,  $\hat{B}_0 \leftarrow 0$ ,  $\hat{b}_0 \leftarrow 0$ 
3:  $\tau = 0$ 
4: for  $i = 1, \dots, n$  do
5:    $\hat{A}_i \leftarrow \hat{A}_{i-1} + \phi(s_i, a_i)\phi^\top(s_i, a_i)$ 
6:    $\hat{B}_i \leftarrow \hat{B}_{i-1} + \phi(s_i, a_i)\phi^\top(s'_i, \pi(s'_i))$ 
7:    $\hat{b}_i \leftarrow \hat{b}_{i-1} + \phi(s_i, a_i)r_i$ 
8:   if  $i$  is a multiple of  $\bar{n}$  then
9:     for  $\tau = \tau, \dots, \tau + N_{\text{upd}} - 1$  do
10:       $\theta_{\tau+1} \leftarrow \theta_\tau + \alpha(\theta'_{\tau+1} - \theta_\tau)$ , where  $\frac{1}{\bar{n}}\hat{A}_i\theta'_{\tau+1} = \gamma\frac{1}{\bar{n}}\hat{B}_i\theta_\tau + \frac{1}{\bar{n}}\hat{b}_i$ 
11: output Q-function  $\hat{Q}^\pi(s, a) = \phi^\top(s, a)\theta_{\tau+1}$ 
```

Algorithm 8. LSPE-Q

Because LSPE-Q must solve the system in (3.25) multiple times, it will require more computational effort than LSTD-Q, which solves the similar system (3.24) only once. On the other hand, the incremental nature of LSPE-Q can offer it advantages over LSTD-Q. For instance, LSPE can benefit from a good initial value of the parameter vector, and better flexibility can be achieved by controlling the step size.

Bellman Residual Minimization

Next, sample-based BRM is briefly discussed. The matrix $C^\top \rho C$ and the vector $C^\top \rho R$ appearing in the idealized BRM equation (3.22) can be estimated from samples. The estimation procedure requires *double* transition samples, that is, for each

state-action sample, two independent transitions to next states must be available. These double transition samples have the form $(s_i, a_i, s'_{i,1}, r_{i,1}, s'_{i,2}, r_{i,2})$, $i = 1, \dots, n$, where $s'_{i,1}$ and $s'_{i,2}$ are independently drawn from the distribution $T(s_i, a_i, \cdot)$, while $r_{i,1} = R(s_i, a_i, s'_{i,1})$ and $r_{i,2} = R(s_i, a_i, s'_{i,2})$. Using these samples, estimates can be built as follows:

$$\begin{aligned}\widehat{(C^\top \rho C)}_i &= \widehat{(C^\top \rho C)}_{i-1} + \\ &\quad [\phi(s_i, a_i) - \gamma\phi(s'_{i,1}, \pi(s'_{i,1}))] \cdot [\phi^\top(s_i, a_i) - \gamma\phi^\top(s'_{i,2}, \pi(s'_{i,2}))] \quad (3.26) \\ \widehat{(C^\top \rho R)}_i &= \widehat{(C^\top \rho R)}_{i-1} + \phi(s_i, a_i) - \gamma\phi(s'_{i,2}, \pi(s'_{i,2}))\end{aligned}$$

starting from zero initial values: $\widehat{(C^\top \rho C)}_0 = 0$, $\widehat{(C^\top \rho R)}_0 = 0$. Once all the samples have been processed, the estimates can be used to approximately solve (3.22), obtaining a parameter vector and thereby an approximate Q-function.

The reason for using double samples is that, if a single sample (s_i, a_i, s'_i, r_i) were used to build sample products of the form $[\phi(s_i, a_i) - \gamma\phi(s'_i, \pi(s'_i))] \cdot [\phi^\top(s_i, a_i) - \gamma\phi^\top(s'_i, \pi(s'_i))]$, such samples would be biased, which in turn would lead to a biased estimate of $C^\top \rho C$ and would make the algorithm unreliable (Baird, 1995; Bertsekas, 1995).

The Need for Exploration

A crucial issue that arises in all the algorithms above is *exploration*: ensuring that the state-action space is sufficiently covered by the available samples. Consider first the exploration of the action space. The algorithms are typically used to evaluate deterministic policies. If samples were only collected according to the current policy π , i.e., if all samples were of the form $(s, \pi(s))$, no information about pairs (s, a) with $a \neq \pi(s)$ would be available. Therefore, the approximate Q-values of such pairs would be poorly estimated and unreliable for policy improvement. To alleviate this problem, exploration is necessary: sometimes, actions different from $\pi(s)$ have to be selected, e.g., in a random fashion. Looking now at the state space, exploration plays another helpful role when samples are collected along trajectories of the system. In the absence of exploration, areas of the state space that are not visited under the current policy would not be represented in the sample set, and the value function would therefore be poorly estimated in these areas, even though they may be important in solving the problem. Instead, exploration drives the system along larger areas of the state space.

Computational Considerations

The computational expense of least-squares algorithms for policy evaluation is typically dominated by solving the linear systems appearing in all of them. However, for one-shot algorithms such as LSTD and BRM, which only need to solve the system

once, the time needed to process samples may become dominant if the number of samples is very large.

The linear systems can be solved in several ways, e.g., by matrix inversion, by Gaussian elimination, or by incrementally computing the inverse with the Sherman-Morrison formula (see Golub and Van Loan, 1996, Chapters 2 and 3). Note also that, when the BF vector ϕ is sparse, as in the often-encountered case of localized BFs, this sparsity can be exploited to greatly improve the computational efficiency of the matrix and vector updates in all the least-squares algorithms.

3.3.4 Bibliographical Notes

The high-level introduction of Section 3.3.1 followed the line of (Farahmand et al, 2009). After that, we followed at places the derivations in Chapter 3 of (Buşoniu et al, 2010a).

LSTD was introduced in the context of V-functions (LSTD-V) by Bradtke and Barto (1996), and theoretically studied by, e.g., Boyan (2002); Konda (2002); Nedić and Bertsekas (2003); Lazaric et al (2010b); Yu (2010). LSTD-Q, the extension to the Q-function case, was introduced by Lagoudakis et al (2002); Lagoudakis and Parr (2003a), who also used it to develop the LSPI algorithm. LSTD-Q was then used and extended in various ways, e.g., by Xu et al (2007); Li et al (2009); Kolter and Ng (2009); Buşoniu et al (2010d,b); Thierry and Scherrer (2010).

LSPE-V was introduced by Bertsekas and Ioffe (1996) and theoretically studied by Nedić and Bertsekas (2003); Bertsekas et al (2004); Yu and Bertsekas (2009). Its extension to Q-functions, LSPE-Q, was employed by, e.g., Jung and Polani (2007a,b); Buşoniu et al (2010d).

The idea of minimizing the Bellman residual was proposed as early as in (Schweitzer and Seidmann, 1985). BRM-Q and variants were studied for instance by Lagoudakis and Parr (2003a); Antos et al (2008); Farahmand et al (2009), while Scherrer (2010) recently compared BRM approaches with projected approaches. It should be noted that the variants of Antos et al (2008) and Farahmand et al (2009), called “modified BRM” by Farahmand et al (2009), eliminate the need for double sampling by introducing a change in the minimization problem (3.9).

3.4 Online Least-Squares Policy Iteration

An important topic in practice is the application of least-squares methods to *online learning*. Unlike in the offline case, where only the final performance matters, in online learning the performance should improve once every few transition samples. Policy iteration can take this requirement into account by performing policy improvements once every few transition samples, before an accurate evaluation of the current policy can be completed. Such a variant is called *optimistic* policy

iteration (Bertsekas and Tsitsiklis, 1996; Sutton, 1988; Tsitsiklis, 2002). In the extreme, fully optimistic case, the policy is improved after every single transition. Optimistic policy updates were combined with LSTD-Q – thereby obtaining optimistic LSPI – in our works (Buşoniu et al, 2010d,b) and with LSPE-Q in (Jung and Polani, 2007a,b). Li et al (2009) explored a non-optimistic, more computationally involved approach to online policy iteration, in which LSPI is fully executed between consecutive sample-collection episodes.

```

1: input BFs  $\phi$ , policy improvement interval  $L$ , exploration schedule,
   initial policy  $\pi_0$ , small constant  $\delta_A > 0$ 
2:  $k \leftarrow 0, t = 0$ 
3:  $(\widehat{A - \gamma B})_0 \leftarrow \delta_A I, \widehat{b}_0 \leftarrow 0$ 
4: observe initial state  $s_0$ 
5: repeat
6:    $a_t \leftarrow \pi_k(s_t) + \text{exploration}$ 
7:   apply  $a_t$ , observe next state  $s_{t+1}$  and reward  $r_{t+1}$ 
8:    $(\widehat{A - \gamma B})_{t+1} \leftarrow (\widehat{A - \gamma B})_t + \phi(s_t, a_t)\phi^\top(s_t, a_t) - \gamma\phi(s_t, a_t)\phi^\top(s_{t+1}, \pi(s_{t+1}))$ 
9:    $\widehat{b}_{t+1} \leftarrow \widehat{b}_t + \phi(s_t, a_t)r_{t+1}$ 
10:  if  $t = (k+1)L$  then
11:    solve  $\frac{1}{t}(\widehat{A - \gamma B})_{t+1}\theta_k = \frac{1}{t}\widehat{b}_{t+1}$  to find  $\theta_k$ 
12:     $\pi_{k+1}(s) \leftarrow \arg \max_{a \in A} \phi^\top(s, a)\theta_k \quad \forall s$ 
13:     $k \leftarrow k + 1$ 
14:   $t \leftarrow t + 1$ 
15: until experiment finished

```

Algorithm 9. Online LSPI

Next, we briefly discuss the online, optimistic LSPI method we introduced in (Buşoniu et al, 2010d), shown here as Algorithm 9. The same matrix and vector estimates are used as in offline LSTD-Q and LSPI, but there are important differences. First, online LSPI collects its own samples, by using its current policy to interact with the system. This immediately implies that exploration must explicitly be added on top of the (deterministic) policy. Second, optimistic policy improvements are performed, that is, the algorithm improves the policy without waiting for the estimates $(\widehat{A - \gamma B})$ and \widehat{b} to get close to their asymptotic values for the current policy. Moreover, these estimates continue to be updated without being reset after the policy changes – so in fact they correspond to multiple policies. The underlying assumption here is that $A - \gamma B$ and b are similar for subsequent policies. A more computationally costly alternative would be to store the samples and rebuild the estimates from scratch before every policy update, but as will be illustrated in the example of Section 3.5, this may not be necessary in practice.

The number L of transitions between consecutive policy improvements is a crucial parameter of the algorithm. For instance, when $L = 1$, online LSPI is fully optimistic. In general, L should not be too large, to avoid potentially bad policies

from being used too long. Note that, as in the offline case, improved policies do not have to be explicitly computed in online LSPI, but can be computed on demand.

3.5 Example: Car on the Hill

In order to exemplify the behavior of least-squares methods for policy iteration, we apply two such methods (offline LSPI and online, optimistic LSPI) to the car-on-the-hill problem (Moore and Atkeson, 1995), a classical benchmark for approximate reinforcement learning. Thanks to its low dimensionality, this problem can be solved using simple linear approximators, in which the BFs are distributed on an equidistant grid. This frees us from the difficulty of customizing the BFs or resorting to a nonparametric approximator, and allows us to focus instead on the behavior of the algorithms.

In the car-on-the-hill problem, a point mass (the ‘car’) must be driven past the top of a frictionless hill by applying a horizontal force, see Figure 3.2, left. For some initial states, due to the limited available force, the car must first be driven to the left, up the opposite slope, and gain momentum prior to accelerating to the right, towards the goal.

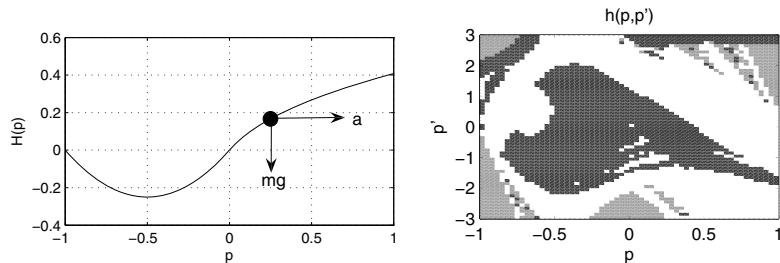


Fig. 3.2 Left: The car on the hill, with the “car” shown as a black bullet. Right: a near-optimal policy (black means $a = -4$, white means $a = +4$, gray means both actions are equally good).

Denoting the horizontal position of the car by p , its dynamics are (in the variant of Ernst et al, 2005):

$$\ddot{p} = \frac{a - 9.81H'(p) - p^2 H'(p) H''(p)}{1 + [H'(p)]^2} \quad (3.27)$$

where the hill shape $H(p)$ is given by $p^2 + p$ for $p < 0$, and by $p^{-1}\sqrt{1+5p^2}$ for $p \geq 0$. The notations \dot{p} and \ddot{p} indicate the first and second time derivatives of p , while $H'(p)$ and $H''(p)$ denote the first and second derivatives of H with respect to p . The state variables are the position and the speed, $s = [p, \dot{p}]^\top$, and the action a is the applied force. Discrete-time dynamics T are obtained by numerically integrating

(3.27) between consecutive time steps, using a discrete time step of 0.1 s. Thus, s_t and p_t are sampled versions of the continuous variables s and p . The state space is $S = [-1,1] \times [-3,3]$ plus a terminal state that is reached whenever s_{t+1} would fall outside these bounds, and the discrete action space is $A = \{-4,4\}$. The goal is to drive past the top of the hill to the right with a speed within the allowed limits, while reaching the terminal state in any other way is considered a failure. To express this goal, the following reward is chosen:

$$R(s_t, a_t, s_{t+1}) = \begin{cases} -1 & \text{if } p_{t+1} < -1 \text{ or } |\dot{p}_{t+1}| > 3 \\ 1 & \text{if } p_{t+1} > 1 \text{ and } |\dot{p}_{t+1}| \leq 3 \\ 0 & \text{otherwise} \end{cases} \quad (3.28)$$

with the discount factor $\gamma = 0.95$. Figure 3.2, right shows a near-optimal policy for this reward function.

To apply the algorithms considered, the Q-function is approximated over the state space using bilinear interpolation on an equidistant 13×13 grid. To represent the Q-function over the discrete action space, separate parameters are stored for the two discrete actions, so the approximator can be written as:

$$\widehat{Q}(s, a_j) = \sum_{i=1}^{169} \varphi_i(s) \theta_{i,j}$$

where the state-dependent BFs $\varphi_i(s)$ provide the interpolation coefficients, with at most 4 BFs being non-zero for any s , and $j = 1, 2$. By replicating the state-dependent BFs for both discrete actions, this approximator can be written in the standard form (3.10), and can therefore be used in least-squares methods for policy iteration.

First, we apply LSPI with this approximator to the car-on-the-hill problem. A set of 10000 random, uniformly distributed state-action samples are independently generated; these samples are reused to evaluate the policy at each policy iteration. With these settings, LSPI typically converges in 7 to 9 iterations (from 20 independent runs, where convergence is considered achieved when the difference between consecutive parameter vectors drops below 0.001). Figure 3.3, top illustrates a subsequence of policies found during a representative run. Subject to the resolution limitations of the chosen representation, a reasonable approximation of the near-optimal policy in Figure 3.2, right is found.

For comparison purposes, we also run an approximate *value iteration* algorithm using the same approximator and the same convergence threshold. (The actual algorithm, called fuzzy Q-iteration, is described in Buşoniu et al (2010c); here, we are only interested in the fact that it is representative for the approximate value iteration class.) The algorithm converges after 45 iterations. This slower convergence of value iteration compared to policy iteration is often observed in practice. Figure 3.3, middle shows a subsequence of policies found by value iteration. Like for the PI algorithms we considered, the policies are implicitly represented by the approximate Q-function, in particular, actions are chosen by maximizing the Q-function as in (3.5). The final solution is different from the one found by LSPI, and the algorithms

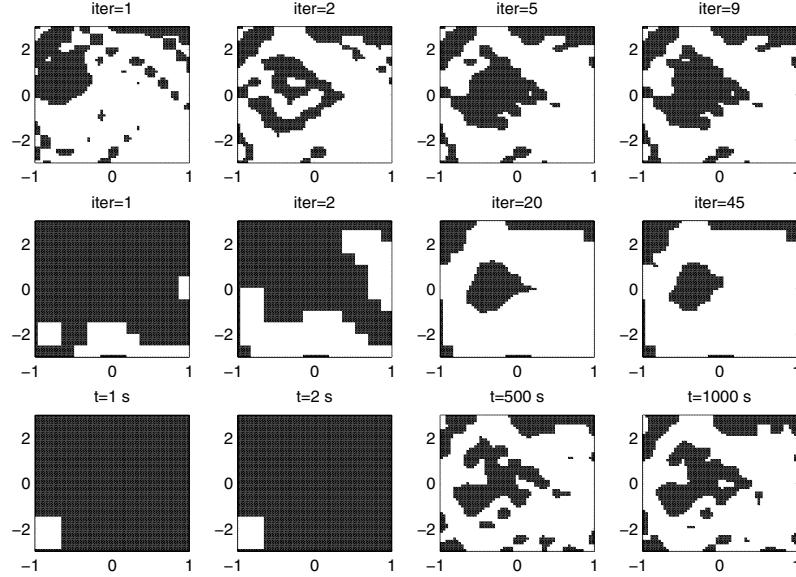


Fig. 3.3 Representative subsequences of policies found by the algorithms considered. Top: offline LSPI; middle: fuzzy Q-iteration; bottom: online LSPI. For axis and color meanings, see Figure 3.2, right, additionally noting that here the negative (black) action is preferred when both actions are equally good.

also converge differently: LSPI initially makes large steps in the policy space at each iteration (so that, e.g., the structure of the policy is already visible after the second iteration), whereas value iteration makes smaller, incremental steps.

Finally, online, optimistic LSPI is applied to the car on the hill. The experiment is run for 1000 s of simulated time, so that in the end 10000 samples have been collected, like for the offline algorithm. This interval is split into separate learning trials, initialized at random initial states and stopping when a terminal state has been reached, or otherwise after 3 s. Policy improvements are performed once every 10 samples (i.e., every 1 s), and an ϵ -greedy exploration strategy is used (see Chapter 2), with $\epsilon = 1$ initially and decaying exponentially so that it reaches a value of 0.1 after 350 s. Figure 3.3, bottom shows a subsequence of policies found during a representative run. Online LSPI makes smaller steps in the policy space than offline LSPI because, in-between consecutive policy improvements, it processes fewer samples, which come from a smaller region of the state space. In fact, at the end of learning LSPI has processed each sample only once, whereas offline LSPI processes all the samples once at every iteration.

Figure 3.4 shows the performance of policies found by online LSPI along the online learning process, in comparison to the final policies found by offline LSPI. The performance is measured by the average empirical return over an equidistant grid of initial states, evaluated by simulation with a precision of 0.001; we call this

average return “score”. Despite theoretical uncertainty about its convergence and near-optimality, online LSPI empirically reaches at least as good performance as the offline algorithm (for encouraging results in several other problems, see our papers (Buşoniu et al, 2010d,b) and Chapter 5 of (Buşoniu et al, 2010a)). For completeness, we also report the score of the – deterministically obtained – value iteration solution: 0.219, slightly lower than that obtained by either version of LSPI.

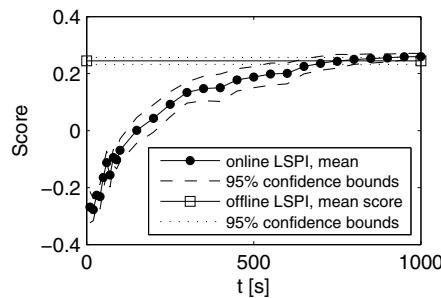


Fig. 3.4 Evolution of the policy score in online LSPI, compared with offline LSPI. Mean values with 95% confidence intervals are reported, from 20 independent experiments.

The execution time was around 34 s for LSPI, around 28 s for online LSPI, and 0.3 s for value iteration. This illustrates the fact that the convergence rate advantage of policy iteration does not necessarily translate into computational savings – since each policy evaluation can have a complexity comparable with the entire value iteration. In our particular case, LSPI requires building the estimates of A , B and b and solving a linear system of equations, whereas for the interpolating approximator employed, each approximate value iteration reduces to a very simple update of the parameters. For other types of approximators, value iteration algorithms will be more computationally intensive, but still tend to require less computation per iteration than PI algorithms. Note also that the execution time of online LSPI is much smaller than the 1000 s *simulated* experiment duration.

3.6 Performance Guarantees

In this section we review the main theoretical guarantees about least-squares methods for policy evaluation and policy iteration. We first discuss convergence properties and the quality of the solutions obtained *asymptotically*, as the number of samples processed and iterations executed grows to infinity. Then, probabilistic bounds are provided on the performance of the policy obtained by using a *finite* number of samples and iterations.

3.6.1 Asymptotic Convergence and Guarantees

While theoretical results are mainly given for V-functions in the literature, they directly extend to Q-functions, by considering the Markov chain of state-action pairs under the current policy, rather than the Markov chain of states as in the case of V-functions. We will therefore use our Q-function-based derivations above to explain and exemplify the guarantees.

Throughout, we require that the BFs are linearly independent, implying that the BF matrix Φ has full column rank. Intuitively, this means there are no redundant BFs.

Projected Policy Evaluation

In the context of projected policy evaluation, one important difference between the LSTD and LSPE families of methods is the following. LSTD-Q will produce a meaningful solution whenever the equation (3.19) has a solution, which can happen for many weight functions ρ . In contrast, to guarantee the convergence of the basic LSPE-Q iteration, one generally must additionally require that the samples follow ρ^π , the *stationary distribution* over state-action pairs induced by the policy considered (stationary distribution of π , for short). Intuitively, this means that the weight of each state-action pair (s, a) is equal to the steady-state probability of this pair along an infinitely-long trajectory generated with the policy π . The projection mapping⁷ Π^{ρ^π} is nonexpansive with respect to the norm $\|\cdot\|_{\rho^\pi}$ weighted by the stationary distribution ρ^π , and because additionally the original Bellman mapping B_Q^π is a contraction with respect to this norm, the projected Bellman mapping $\Pi^{\rho^\pi}(B_Q^\pi(\cdot))$ is also a contraction.⁸

Confirming the LSTD is not very dependent on using ρ^π , Yu (2010) proved that with a minor modification, the solution found by LSTD converges as $n \rightarrow \infty$, even when one policy is evaluated using samples from a different policy – the so-called off-policy case. Even for LSPE, it may be possible to mitigate the destabilizing effects of violating convergence assumptions, by controlling the step size α . Furthermore, a modified LSPE-like update has recently been proposed that converges without requiring that $\Pi^{\rho}(B_Q^\pi(\cdot))$ is a contraction, see Bertsekas (2011a,b). These types of results are important in practice because they pave the way for *reusing* samples to evaluate different policies, that is, at different iterations of the overall PI algorithm. In contrast, if the stationary distribution must be followed, new samples have to be generated at each iteration, using the current policy.

⁷ A projection mapping Π^{ρ^π} applied to a function f w.r.t. a space \mathcal{F} returns the closest element in \mathcal{F} to the function f , where the distance is defined according to the L2 norm and the measure ρ^π .

⁸ A mapping $f(x)$ is a contraction with factor $\gamma < 1$ if for any x, x' , $\|f(x) - f(x')\| \leq \gamma \|x - x'\|$. The mapping is a nonexpansion (a weaker property) if the inequality holds for $\gamma = 1$.

Assuming now, for simplicity, that the stationary distribution ρ^π is used, and thus that the projected Bellman equation (3.19) has a unique solution (the projected Bellman operator is a contraction and it admits one unique fixed point), the following informal, but intuitive line of reasoning is useful to understand the convergence of the sample-based LSTD-Q and LSPE-Q. Asymptotically, as $n \rightarrow \infty$, it is true that $\frac{1}{n}\hat{A}_n \rightarrow A$, $\frac{1}{n}\hat{B}_n \rightarrow B$, and $\frac{1}{n}\hat{b}_n \rightarrow b$, because the empirical distribution of the state-action samples converges to ρ , while the empirical distribution of next-state samples s' for each pair (s, a) converges to $T(s, a, s')$. Therefore, the practical, sample-based LSTD-Q converges to its idealized version (3.19), and LSTD-Q asymptotically finds the solution of the projected Bellman equation. Similarly, the sample-based LSPE-Q asymptotically becomes equivalent to its idealized version (3.20), which is just an incremental variant of (3.19) and will therefore produce the same solution in the end. In fact, it can additionally be shown that, as n grows, the solutions of LSTD-Q and LSPE-Q converge to each other faster than they converge to their limit, see Yu and Bertsekas (2009).

Let us investigate now the quality of the solution. Under the stationary distribution ρ^π , we have (Bertsekas, 2011a; Tsitsiklis and Van Roy, 1997):

$$\|Q^\pi - \hat{Q}^\pi\|_{\rho^\pi} \leq \frac{1}{\sqrt{1-\gamma^2}} \|Q^\pi - \Pi^{\rho^\pi}(Q^\pi)\|_{\rho^\pi} \quad (3.29)$$

where \hat{Q}^π is the Q-function given by the parameter θ that solves the projected Bellman equation (3.19) for $\rho = \rho^\pi$. Thus, we describe the representation power of the approximator by the distance $\|Q^\pi - \Pi^{\rho^\pi}(Q^\pi)\|_{\rho^\pi}$ between the true Q-function Q^π and its projection $\Pi^{\rho^\pi}(Q^\pi)$. As the approximator becomes more powerful, this distance decreases. Then, projected policy evaluation leads to an approximate Q-function \hat{Q}^π with an error proportional to this distance. The proportion is given by the discount factor γ , and grows as γ approaches 1. Recently, efforts have been made to refine this result in terms of properties of the dynamics and of the set $\hat{\mathcal{Q}}$ of representable Q-functions (Scherrer, 2010; Yu and Bertsekas, 2010).

Bellman Residual Minimization

The following relationship holds for any Q-function Q , see e.g. Scherrer (2010):

$$\|Q^\pi - Q\|_{\rho^\pi} \leq \frac{1}{1-\gamma} \|Q - B_Q^\pi(Q)\|_{\rho^\pi} \quad (3.30)$$

and with ρ^π the stationary distribution. Consider now the on-policy BRM solution – the Q-function \hat{Q}^π given by the parameter that solves the BRM equation (3.22) for $\rho = \rho^\pi$. Because this Q-function minimizes the right-hand side of the inequality (3.30), the error $\|Q^\pi - \hat{Q}^\pi\|_{\rho^\pi}$ of the solution found is also small.

Comparing projected policy evaluation with BRM, no general statements can be made about the relative quality of their solution. For instance, in the context of policy evaluation only, Scherrer (2010) suggests based on an empirical study that projected policy evaluation may outperform BRM more often (for more problem instances) than the other way around; but when it fails, it may do so with much larger errors than those of BRM. For additional insight and comparisons, see, e.g., Munos (2003); Scherrer (2010).

Approximate Policy Iteration

A general result about policy iteration can be given in terms of the infinity norm, as follows. If the policy evaluation error $\|\hat{Q}^{\pi_k} - Q^{\pi_k}\|_{\infty}$ is upper-bounded by ε at every iteration $k \geq 0$ (see again Algorithm 6), and if policy improvements are exact (according to our assumptions in Section 3.2), then policy iteration eventually produces policies with a performance (i.e., the corresponding value function) that lies within a bounded distance from the optimal performance (Bertsekas and Tsitsiklis, 1996; Lagoudakis and Parr, 2003a) (i.e., the optimal value function):

$$\limsup_{k \rightarrow \infty} \|Q^{\pi_k} - Q^*\|_{\infty} \leq \frac{2\gamma}{(1-\gamma)^2} \cdot \varepsilon \quad (3.31)$$

Here, Q^* is the optimal Q-function and corresponds to the optimal performance, see Chapter 2. Note that if approximate policy improvements are performed, a similar bound holds, but the policy improvement error must also be included in the right hand side.

An important remark is that the sequence of policies is generally not guaranteed to converge to a fixed policy. For example, the policy may end up oscillating along a limit cycle. Nevertheless, all the policies along the cycle will have a high performance, in the sense of (3.31).

Note that (3.31) uses infinity norms, whereas the bounds (3.29) and (3.30) for the policy evaluation component use Euclidean norms. The two types of bounds cannot be easily combined to yield an overall bound for approximate policy iteration. Policy iteration bounds for Euclidean norms, which we will not detail here, were developed by Munos (2003).

Consider now optimistic variants of policy iteration, such as online LSPI. The performance guarantees above rely on small policy evaluation errors, whereas in the optimistic case, the policy is improved before an accurate value function is available, which means the policy evaluation error can be very large. For this reason, the behavior of optimistic policy iteration is theoretically poorly understood at the moment, although the algorithms often work well in practice. See Bertsekas (2011a) and Section 6.4 of Bertsekas and Tsitsiklis (1996) for discussions of the difficulties involved.

3.6.2 Finite-Sample Guarantees

The results reported in the previous section analyze the asymptotic performance of policy evaluation methods when the number of samples tends to infinity. Nonetheless, they do not provide any guarantee about how the algorithms behave when only a finite number of samples is available. In this section we report recent finite-sample bounds for LSTD and BRM and we discuss how they propagate through iterations in the policy iteration scheme.

While in the previous sections we focused on algorithms for the approximation of Q-functions, for sake of simplicity we report here the analysis for V-function approximation. The notation and the setting is exactly the same as in Section 3.3.2 and we simply redefine it for V-functions. We use a linear approximation architecture with parameters $\theta \in \mathbb{R}^d$ and basis functions $\varphi_i, i = 1, \dots, d$ now defined as a mapping from the state space S to \mathbb{R} . We denote by $\phi : S \rightarrow \mathbb{R}^d$, $\phi(\cdot) = [\varphi_1(\cdot), \dots, \varphi_d(\cdot)]^\top$ the BF vector (feature vector), and by \mathcal{F} the linear function space spanned by the BFs φ_i , that is $\mathcal{F} = \{f_\theta | \theta \in \mathbb{R}^d \text{ and } f_\theta(\cdot) = \phi^\top(\cdot)\theta\}$. We define $\tilde{\mathcal{F}}$ as the space obtained by truncating the functions in \mathcal{F} at V_{\max} (recall that V_{\max} gives the maximum return and upper-bounds any value function). The truncated function \tilde{f} is equal to $f(s)$ in all the states where $|f(s)| \leq V_{\max}$ and it is equal to $\text{sgn}(f(s))V_{\max}$ otherwise. Furthermore, let L be an upper bound for all the BFs, i.e., $\|\varphi_i\|_\infty \leq L$ for $i = 1, \dots, d$. In the following we report the finite-sample analysis of the policy evaluation performance of LSTD-V and BRM-V, followed by the analysis of policy iteration algorithms that use LSTD-V and BRM-V in the policy evaluation step. The truncation to V_{\max} is used for LSTD-V, but not for BRM-V.

Pathwise LSTD

Let π be the current policy and V^π its V-function. Let (s_t, r_t) with $t = 1, \dots, n$ be a sample path (trajectory) of size n generated by following the policy π and $\Phi = [\phi^\top(s_1); \dots; \phi^\top(s_n)]$ be the BF matrix defined at the encountered states, where “;” denotes a vertical stacking of the vectors $\phi^\top(s_t)$ in the matrix. Pathwise LSTD-V is a version of LSTD-V obtained by defining an empirical transition matrix \hat{T} as follows: $\hat{T}_{ij} = 1$ if $j = i + 1, j \neq n$, otherwise $\hat{T}_{ij} = 0$. When applied to a vector $s = [s_1, \dots, s_n]^\top$, this transition matrix returns $(\hat{T}s)_t = s_{t+1}$ for $1 \leq t < n$ and $(\hat{T}s)_n = 0$. The rest of the algorithm exactly matches the standard LSTD and returns a vector θ as the solution of the linear system $A\theta = \gamma B\theta + b$, where $A = \Phi^\top \Phi$, $B = \gamma \Phi^\top \hat{T} \Phi$, and $b = \Phi^\top R$. Similar to the arguments used in Section 3.6.1, it is easy to verify that the empirical transition matrix \hat{T} results in an empirical Bellman operator which is a contraction and thus that the previous system always admits at least one solution. Although there exists a unique fixed point, there might be multiple solutions θ . In the following, we use $\hat{\theta}$ to denote the solution with minimal norm, that is

$\hat{\theta} = (A - \gamma B)^+ b$, where $(A - \gamma B)^+$ is the Moore-Penrose pseudo-inverse of the matrix $A - \gamma B$.⁹

For the pathwise LSTD-V algorithm the following performance bound has been derived in (Lazaric et al, 2010b).

Theorem 3.1. (Pathwise LSTD-V) Let $\omega > 0$ be the smallest eigenvalue of the Gram matrix $G \in \mathbb{R}^{d \times d}$, $G_{ij} = \int \phi_i^\top(x) \phi_j(x) \rho^\pi(dx)$. Let assume that the policy π induces a stationary β -mixing process (Meyn and Tweedie, 1993) on the MDP at hand with a stationary distribution ρ^π ¹⁰. Let (s_t, r_t) with $t = 1, \dots, n$ be a path generated by following policy π for $n > n^\pi(\omega, \delta)$ steps, where $n^\pi(\omega, \delta)$ is a suitable number of steps depending on parameters ω and δ . Let $\hat{\theta}$ be the pathwise LSTD-V solution and $\tilde{f}_{\hat{\theta}}$ be the truncation of its corresponding function, then:¹¹

$$\begin{aligned} \|\tilde{f}_{\hat{\theta}} - V^\pi\|_{\rho^\pi} &\leq \frac{2}{\sqrt{1-\gamma^2}} \left(2\sqrt{2} \|V^\pi - \Pi^{\rho^\pi} V^\pi\|_{\rho^\pi} + \tilde{O} \left(L \|\theta^*\| \sqrt{\frac{d \log 1/\delta}{n}} \right) \right) \\ &\quad + \frac{1}{1-\gamma} \tilde{O} \left(V_{\max} L \sqrt{\frac{d \log 1/\delta}{\omega n}} \right) \end{aligned} \quad (3.32)$$

with probability $1 - \delta$, where θ^* is such that $f_{\theta^*} = \Pi^{\rho^\pi} V^\pi$.

In the following we analyze the main terms of the previous theorem.

- $\|V^\pi - \Pi^{\rho^\pi} V^\pi\|_{\rho^\pi}$: this term is the *approximation error* and it only depends on the target function V^π and how well the functions in \mathcal{F} can approximate it. This can be made small whenever some prior knowledge about V^π is available and the BFs are carefully designed. Furthermore, we expect it to decrease with the number d of BFs but at the cost of increasing the other terms in the bound. As it can be noticed, when n goes to infinity, the remaining term in the bound is $\frac{4\sqrt{2}}{\sqrt{1-\gamma^2}} \|V^\pi - \Pi^{\rho^\pi} V^\pi\|_{\rho^\pi}$ which matches the asymptotic performance bound (3.29) of LSTD up to constants.
- $\tilde{O} \left(V_{\max} L \sqrt{\frac{d \log 1/\delta}{\omega n}} \right)$: this is the first *estimation error* and it is due to bias introduced by the fact that the pathwise LSTD-V solution is computed on a finite number of random samples. As it can be noticed, this term decreases with the number of samples as $n^{-1/2}$ but it also depends on the number of BFs d in \mathcal{F} , the range of the BFs L , and the range of the V-functions on the MDP at hand. The term also shows a critical dependency on the smallest eigenvalue ω of the model-based Gram matrix G . Whenever the BFs are linearly independent

⁹ Note that whenever a generic square matrix D is invertible $D^+ = D^{-1}$.

¹⁰ Roughly speaking, a fast mixing process is such that starting from an arbitrary initial distribution and following the current policy, the state probability distribution rapidly tends to the stationary distribution of the Markov chain.

¹¹ For simplicity, in the \tilde{O} terms we omit constants, logarithmic factors in d and n , and the dependency on the characteristic parameters of the β -mixing process.

and the matrix G is well-conditioned under the stationary distribution ρ^π , ω is guaranteed to be far from 0.

- $\tilde{O}\left(L\|\theta^*\|\sqrt{\frac{d \log 1/\delta}{n}}\right)$: this is the second *estimation error* and similarly to the previous one, it decreases with the number of samples. This estimation error does not depend on the smallest eigenvalue ω but it has an additional dependency on $\|\theta^*\|$ which in some cases might add a further dependency on the dimensionality of \mathcal{F} and could be large whenever the BFs are not linear independent under the stationary distribution ρ^π .

The previous bound gives a better understanding of what are the most relevant terms determining the quality of the approximation of LSTD (most notably some properties of the function space). Furthermore, the bound can also be used to have a rough estimate of the number of samples needed to achieve a certain desired approximation quality. For instance, let us assume the approximation error is zero and an error level ϵ is desired (i.e., $\|\tilde{f}_{\hat{\theta}} - V^\pi\|_{\rho^\pi}$). In this case, the bound suggests that n should be bigger than $(1-\gamma)^{-2}V_{\max}^2L^2d/\epsilon^2$ (where here we ignore the terms that are not available at design time such as $\|\theta^*\|$ and ω).

Pathwise LSTD-V strictly requires the samples to be generated by following the current policy π (i.e., it is *on-policy*) and the finite-sample analysis bounds its performance according to the stationary distribution ρ^π . Although an asymptotic analysis for off-policy LSTD exists (see Section 3.6.1), to the best of our knowledge no finite-sample result is available for off-policy LSTD. The closest result has been obtained for the modified Bellman residual algorithm introduced by Antos et al (2008), which is an off-policy algorithm which reduces to LSTD when linear spaces are used. Nonetheless, the finite-sample analysis reported by Antos et al (2008) does not extend from bounded to linear spaces and how to bound the performance of off-policy LSTD is still an open question.

Bellman Residual Minimization

We now consider the BRM-V algorithm, which finds V-functions. Similar to BRM-Q (see Section 3.3.2) n samples $(s_i, s'_{i,1}, r_{i,1}, s'_{i,2}, r_{i,2})$ are available, where s_i are drawn i.i.d. from an arbitrary distribution μ , $(s'_{i,1}, r_{i,1})$ and $(s'_{i,2}, r_{i,2})$ are two independent samples drawn from $T(s_i, \pi(s_i), \cdot)$, and $r_{i,1}, r_{i,2}$ are the corresponding rewards. The algorithm works similarly to BRM-Q but now the BFs are functions of the state only. Before reporting the bound on the performance of BRM-V, we introduce:

$$C^\pi(\mu) = (1-\gamma)\|(I - \gamma P^\pi)^{-1}\|_\mu, \quad (3.33)$$

which is related to the concentrability coefficient (see e.g. Antos et al (2008)) of the discounted future state distribution starting from μ and following policy π , i.e., $(1-\gamma)\mu(I - \gamma^\pi)^{-1}$ w.r.t. μ . Note that if the discounted future state distribution is not absolutely continuous w.r.t. μ , then $C^\pi(\mu) = \infty$. We can now report the bound derived by Maillard et al (2010).

Theorem 3.2. (BRM-V) Let $n \geq n^\pi(\omega, \delta)$ be the number of samples drawn i.i.d. according to an arbitrary distribution μ and ω be the smallest eigenvalue of the model-based Gram matrix $G \in \mathbb{R}^{d \times d}$, $G_{ij} = \int \phi_i^\top(x) \phi_j(x) \mu(dx)$. If $\hat{\theta}$ is the BRM-V solution and $f_{\hat{\theta}}$ the corresponding function, then the Bellman residual is bounded as:

$$\|f_{\hat{\theta}} - B_V^\pi f_{\hat{\theta}}\|_\mu^2 \leq \inf_{f \in \mathcal{F}} \|f - B_V^\pi f\|_\mu^2 + \tilde{O}\left(\frac{1}{\xi_\pi^2} L^4 R_{\max}^2 \sqrt{\frac{d \log 1/\delta}{n}}\right),$$

with probability $1 - \delta$, where $\xi_\pi = \frac{\omega(1-\gamma)^2}{C^\pi(\mu)^2}$. Furthermore, the approximation error of V^π is:

$$\|V^\pi - f_{\hat{\theta}}\|_\mu^2 \leq \left(\frac{C^\pi(\mu)}{1-\gamma}\right)^2 \left((1+\gamma\|P^\pi\|_\mu)^2 \inf_{f \in \mathcal{F}} \|V^\pi - f\|_\mu^2 + \tilde{O}\left(\frac{1}{\xi_\pi^2} L^4 R_{\max}^2 \sqrt{\frac{d \log 1/\delta}{n}}\right)\right).$$

We analyze the first statement of the previous theorem.

- $\|f_{\hat{\theta}} - B_V^\pi f_{\hat{\theta}}\|_\mu^2$: the left-hand side of the bound is the exact Bellman residual of the BRM-V solution $f_{\hat{\theta}}$. This result is important to show that minimizing the empirical Bellman residual (i.e., the one computed using only the two independent samples $(s'_{i,1}, r_{i,1})$ and $(s'_{i,2}, r_{i,2})$) on a finite number of states s_i leads to a small exact Bellman residual on the whole state space S .
- $\inf_{f \in \mathcal{F}} \|f - B_V^\pi f\|_\mu^2$: this term is the smallest Bellman residual that can be achieved with functions in the space \mathcal{F} and it plays the role of the *approximation error*. Although difficult to analyze, it is possible to show that this term can be small for the specific class of MDPs where both the reward function and transition kernel are Lipschitz (further discussion can be found in (Munos and Szepesvári, 2008)).
- $\tilde{O}\left(\frac{1}{\xi_\pi^2} L^4 R_{\max}^2 \sqrt{\frac{d \log 1/\delta}{n}}\right)$: this is the *estimation error*. As it can be noticed, unlike in LSTD bounds, here the *squared* loss is bounded and the estimation error decreases with a slower rate of $O(n^{-1/2})$. Beyond the dependency on some terms characteristic of the MDP and the BFs, this term also depends on the inverse of the smallest eigenvalue of the Gram matrix computed w.r.t. the sampling distribution μ . Although this dependency could be critical, ω can be made big with a careful design of the distribution μ and the BFs.

The second statement does not introduce additional relevant terms in the bound. We postpone the discussion about the term $C^\pi(\mu)$ and a more detailed comparison to LSTD to the next section.

Policy Iteration

Up until now, we reported the performance bounds for pathwise-LSTD-V and BRM-V for policy evaluation. Next, we provide finite-sample analysis for *policy*

iteration algorithms, LSPI and BRM-PI, in which at each iteration k , pathwise-LSTD-V and, respectively, BRM-V are used to compute an approximation of V^{π_k} . In particular, we report performance bounds by comparing the value of the policy returned by these algorithms after K iterations, V^{π_K} , and the optimal V-function, V^* . In order to derive performance bounds for LSPI and BRM-PI, one needs to first relate the error after K iterations, $\|V^* - V^{\pi_K}\|$, to the one at each iteration k of these algorithms, and then replace the error at each iteration with the policy evaluation bounds of Theorems 3.1 and 3.2. Antos et al (2008) (Lemma 12) derived the following bound that relates $\|V^* - V^{\pi_K}\|_\sigma$ to the error at each iteration k

$$\|V^* - V^{\pi_K}\|_\sigma \leq \frac{4\gamma}{(1-\gamma)^2} \left(\sqrt{C_{\sigma,v}} \max_{0 \leq k < K} \|V_k - B_V^{\pi_k} V_k\|_v + \gamma^{\frac{K-1}{2}} R_{\max} \right), \quad (3.34)$$

where V_k is the approximation returned by the algorithm of the value function V^{π_k} at each iteration, σ and v are any measures over the state space, and $C_{\sigma,v}$ is the concentrability term from Definition 2 of Antos et al (2008). It is important to note that there is no assumption in this bound on how the sequence V_k is generated. Before reporting the bounds for pathwise-LSPI and BRM-PI, it is necessary to make the following assumptions.

Assumption 1 (Uniform stochasticity). *Let μ be a distribution over the state space. There exists a constant C , such that for all $s \in S$ and for all $a \in A$, $T(s,a,\cdot)$ is uniformly dominated by $\mu(\cdot)$, i.e., $T(s,a,\cdot) \leq C\mu(\cdot)$.*

This assumption guarantees that there is not state-action pair whose transition to the next state is too much concentrated to a limited set of states. It can be easily shown that under this assumption, $C^\pi(\mu)$ defined in Eq. 3.33 and the concentrability term $C_{\sigma,\mu}$, for any σ , are both upper-bounded by C . We may now report a bound for BRM-PI when Assumption 1 holds for the distribution μ used at each iteration of the algorithm.

Theorem 3.3. *For any $\delta > 0$, whenever $n \geq n(\delta/K)$ with $n(\delta/K)$ is a suitable number of samples, with probability $1 - \delta$, the empirical Bellman residual minimizer f_{θ_k} exists for all iterations $1 \leq k < K$, thus the BRM-PI algorithm is well defined, and the performance V^{π_K} of the policy π_K returned by the algorithm is such that:*

$$\|V^* - V^{\pi_K}\|_\infty \leq \tilde{O} \left(\frac{1}{(1-\gamma)^2} \left(C^{3/2} E_{BRM}(\mathcal{F}) + \frac{\sqrt{C}}{\xi} \left(\frac{d \log(K/\delta)}{n} \right)^{1/4} + \gamma^{K/2} \right) \right),$$

where $E_{BRM}(\mathcal{F}) = \sup_{\pi \in \mathcal{G}(\mathcal{F})} \inf_{f \in \mathcal{F}} \|f - V^\pi\|_\mu$, $\mathcal{G}(\mathcal{F})$ is the set of all greedy policies w.r.t. the functions in \mathcal{F} , and $\xi = \frac{\omega(1-\gamma)^2}{C^2} \leq \xi_\pi$, $\pi \in \mathcal{G}(\mathcal{F})$.

In this theorem, $E_{BRM}(\mathcal{F})$ looks at the smallest approximation error for the V-function of a policy π , and takes the worst case of this error across the set of policies greedy in some approximate V-function from \mathcal{F} . The second term is the estimation error and it contains the same terms as in Theorem 3.2 and it decreases as $\tilde{O}(d/n)$.

Finally, the last term $\gamma^{K/2}$ simply accounts for the error due to the finite number of iterations and it rapidly goes to zero as K increases.

Now we turn to pathwise-LSPI and report a performance bound for this algorithm. At each iteration k of pathwise-LSPI, samples are collected by following a single trajectory generated by the policy under evaluation, π_k , and pathwise-LSTD is used to compute an approximation of V^{π_k} . In order to plug in the pathwise-LSTD bound (Theorem 3.1) in Eq. 3.34, one should first note that $\|V_k - B_V^{\pi_k} V_k\|_{\rho^{\pi_k}} \leq (1 + \gamma) \|V_k - V^{\pi_k}\|_{\rho^{\pi_k}}$. This way instead of the Bellman residual, we bound the performance of the algorithm by using the approximation error $V_k - V^{\pi_k}$ at each iteration. It is also important to note that the pathwise-LSTD bound requires the samples to be collected by following the policy under evaluation. This might introduce severe problems in the sequence of iterations. In fact, if a policy concentrates too much on a small portion of the state space, even if the policy evaluation is accurate on those states, it might be arbitrary bad on the states which are not covered by the current policy. As a result, the policy improvement is likely to generate a bad policy which could, in turn, lead to an arbitrary bad policy iteration process. Thus, the following assumption needs to be made here.

Assumption 2 (Lower-bounding distribution). *Let μ be a distribution over the state space. For any policy π that is greedy w.r.t. a function in the truncated space $\tilde{\mathcal{F}}$, $\mu(\cdot) \leq \kappa \rho^\pi(\cdot)$, where $\kappa < \infty$ is a constant and ρ^π is the stationary distribution of policy π .*

It is also necessary to guarantee that with high probability a unique pathwise-LSTD solution exists at each iteration of the pathwise-LSPI algorithm, thus, the following assumption is needed.

Assumption 3 (Linear independent BFs). *Let μ be the lower-bounding distribution from Assumption 2. We assume that the BFs $\phi(\cdot)$ of the function space $\tilde{\mathcal{F}}$ are linearly independent w.r.t. μ . In this case, the smallest eigenvalue ω_μ of the Gram matrix $G_\mu \in \mathbb{R}^{d \times d}$ w.r.t. μ is strictly positive.*

Assumptions 2 and 3 (plus some minor assumptions on the characteristic parameters of the β -mixing processes observed during the execution of the algorithm) are sufficient to have a performance bound for pathwise-LSPI. However, in order to make it easier to compare the bound with the one for BRM-PI, we also assume that Assumption 1 holds for pathwise-LSPI.

Theorem 3.4. *Let us assume that at each iteration k of the pathwise-LSPI algorithm, a path of size $n > n(\omega_\mu, \delta)$ is generated from the stationary β -mixing process with stationary distribution $\rho_{k-1} = \rho^{\pi_{k-1}}$. Let $V_{-1} \in \tilde{\mathcal{F}}$ be an arbitrary initial V-function, V_0, \dots, V_{K-1} ($\tilde{V}_0, \dots, \tilde{V}_{K-1}$) be the sequence of V-functions (truncated V-functions) generated by pathwise-LSPI after K iterations, and π_K be the greedy policy w.r.t. the truncated V-function \tilde{V}_{K-1} . Then under Assumptions 1–3 and some*

assumptions on the characteristic parameters of the β -mixing processes observed during the execution of the algorithm, with probability $1 - \delta$, we have:

$$\|V^* - V^{\pi_K}\|_\infty \leq \tilde{O} \left(\frac{1}{(1-\gamma)^2} \left(\frac{\sqrt{C\kappa}}{1-\gamma} E_{LSTD}(\mathcal{F}) + \frac{\kappa\sqrt{C}}{\sqrt{\omega_\mu}} \left(\frac{d \log(K/\delta)}{n} \right)^{1/2} + \gamma^{K/2} \right) \right),$$

where $E_{LSTD}(\mathcal{F}) = \sup_{\pi \in \mathcal{G}(\tilde{\mathcal{F}})} \inf_{f \in \mathcal{F}} \|f - V^\pi\|_{\rho^\pi}$ and $\mathcal{G}(\tilde{\mathcal{F}})$ is the set of all greedy policies w.r.t. the functions in $\tilde{\mathcal{F}}$.

Note that the initial policy π_0 is greedy in the V-function V_{-1} , rather than being arbitrary; that is why we use the index -1 for the initial V-function.

Comparing the performance bounds of BRM-PI and pathwise-LSPI we first notice that BRM-PI has a poorer estimation rate of $O(n^{-1/4})$ instead of $O(n^{-1/2})$. We may also see that the approximation error term in BRM-PI, $E_{BRM}(\mathcal{F})$, is less complex than that for pathwise-LSPI, $E_{LSTD}(\mathcal{F})$, as the norm in $E_{BRM}(\mathcal{F})$ is only w.r.t. the distribution μ while the one in $E_{LSTD}(\mathcal{F})$ is w.r.t. the stationary distribution of any policy in $\mathcal{G}(\tilde{\mathcal{F}})$. The assumptions used by the algorithms are also different. In BRM-PI, it is assumed that a generative model is available, and thus, the performance bounds may be obtained under any sampling distribution μ , specifically the one for which Assumption 1 holds. On the other hand, at each iteration of pathwise-LSPI, it is required to use a single trajectory generated by following the policy under evaluation. This can provide performance bounds only under the stationary distribution of that policy, and accurately approximating the current policy under the stationary distribution may not be enough in a policy iteration scheme, because the greedy policy w.r.t. that approximation may be arbitrarily poor. Therefore, we may conclude that the performance of BRM-PI is better controlled than that of pathwise-LSPI. This is reflected in the fact that the concentrability terms may be controlled in the BRM-PI by only choosing a uniformly dominating distribution μ (Assumption 1), such as a uniform distribution, while in pathwise-LSPI, we are required to make stronger assumptions on the stationary distributions of the policies encountered at the iterations of the algorithm, such as being lower-bounded by a uniform distribution (Assumption 2).

3.7 Further Reading

Many extensions and variations of the methods introduced in Section 3.3 have been proposed, and we unfortunately do not have the space to describe them all. Instead, in the present section, we will (non-exhaustively) touch upon some of the highlights in this active field of research, providing pointers to the literature for the reader interested in more details.

As previously mentioned in Footnote 3, variants of approximate policy evaluation that employ a *multistep* Bellman mapping can be used. This mapping is parameterized by $\lambda \in [0, 1]$, and is given, e.g., in the case of Q-functions by:

$$B_{Q,\lambda}^{\pi}(Q) = (1 - \lambda) \sum_{t=0}^{\infty} \lambda^t (B_Q^{\pi})^{t+1}(Q)$$

where $(B_Q^{\pi})^t$ denotes the t -times composition of B_Q^{π} with itself. In this chapter, we only considered the single-step case, in which $\lambda = 0$, but in fact approximate policy evaluation is often discussed in the general $\lambda \in [0, 1]$ case, see e.g. Nedić and Bertsekas (2003); Yu (2010); Bertsekas and Ioffe (1996); Yu and Bertsekas (2009) and also the discussion of the so-called TD(λ) algorithm in Chapter 2. Note that in the original LSPI, a nonzero λ would prevent sample reuse; instead, at every iteration, new samples would have to be generated with the current policy.

Nonparametric approximators are not predefined, but are automatically constructed from the data, so to a large extent they free the user from the difficult task of designing the BFs. A prominent class of nonparametric techniques is kernel-based approximation, which was combined, e.g., with LSTD by Xu et al (2005, 2007); Jung and Polani (2007b); Farahmand et al (2009), with LSPE by Jung and Polani (2007a,b), and with BRM by Farahmand et al (2009). The related framework of Gaussian processes has also been used in policy evaluation (Rasmussen and Kuss, 2004; Engel et al, 2005; Taylor and Parr, 2009). In their basic form, the computational demands of kernel-based methods and Gaussian processes grow with the number of samples considered. Since this number can be large in practice, many of the approaches mentioned above employ kernel sparsification techniques to limit the number of samples that contribute to the solution (Xu et al, 2007; Engel et al, 2003, 2005; Jung and Polani, 2007a,b).

Closely related to sparsification is the technique of *regularization*, which controls the complexity of the solution (Farahmand et al, 2009; Kolter and Ng, 2009). For instance, to obtain a regularized variant of LSTD, a penalty term can be added to the projected policy evaluation problem (3.8) to obtain:

$$\min_{\hat{Q} \in \hat{\mathcal{Q}}} \left[\left\| \hat{Q} - \Pi(B_Q^{\pi}(\hat{Q})) \right\| + \beta v(\hat{Q}) \right]$$

where β is a positive regularization coefficient and v is a penalty function that grows with the complexity of \hat{Q} (e.g., a norm of the approximator's parameter vector). Note that Farahmand et al (2009) also add a similar regularization term to the projection Π , and additionally discusses a regularized version of BRM.

In an effort to reduce the computational costs of LSTD, so-called *incremental* variants have been proposed, in which only a few of the parameters are updated at a given iteration (Geramifard et al, 2006, 2007).

A generalized, *scaled* variant of LSPE, discussed e.g. by Bertsekas (2011a), can be obtained by first rewriting the LSPE update (3.20) into the equivalent form (assuming A is invertible):

$$\theta_{\tau+1} = \theta_{\tau} - \alpha A^{-1} [(A - \gamma B) \theta_{\tau} - b]$$

and then replacing A^{-1} by a more general scaling matrix Γ :

$$\theta_{\tau+1} = \theta_\tau - \alpha\Gamma[(A - \gamma B)\theta_\tau - b]$$

For appropriate choices of Γ and α , this algorithm converges under more general conditions than the original LSPE.

In the context of BRM, Antos et al (2008) eliminated the requirement of double sampling by modifying the minimization problem solved by BRM. They showed that single-sample estimates of the coefficients in this modified problem are unbiased, while the solution stays meaningful.

We also note active research into alternatives to least-squares methods for policy evaluation and iteration, in particular, techniques based on gradient updates (Sutton et al, 2009b,a; Maei et al, 2010) and on Monte Carlo simulations (Lagoudakis and Parr, 2003b; Dimitrakakis and Lagoudakis, 2008; Lazaric et al, 2010a).

While an extensive tutorial such as this one, focusing on a unified view and theoretical study of policy iteration with LSTD, LSPE, and BRM policy evaluation, has not been available in the literature until now, these methods have been treated in various recent books and surveys on reinforcement learning and dynamic programming. For example, the interested reader should know that Chapter 3 of (Buşoniu et al, 2010a) describes LSTD-Q and LSPE-Q as part of an introduction to approximate reinforcement learning and dynamic programming, that (Munos, 2010) touches upon LSTD and BRM, that Chapter 3 of (Szepesvári, 2010) outlines LSTD and LSPE methods, and that (Bertsekas, 2011a, 2010) concern to a large extent these two types of methods.

Chapter 8 of this book presents a more general view over the field of approximate reinforcement learning, without focusing on least-squares methods.

References

- Antos, A., Szepesvári, C., Munos, R.: Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning* 71(1), 89–129 (2008)
- Baird, L.: Residual algorithms: Reinforcement learning with function approximation. In: Proceedings 12th International Conference on Machine Learning (ICML-1995), Tahoe City, U.S, pp. 30–37 (1995)
- Bertsekas, D.P.: A counterexample to temporal differences learning. *Neural Computation* 7, 270–279 (1995)
- Bertsekas, D.P.: Approximate dynamic programming. In: *Dynamic Programming and Optimal Control*, Ch. 6, vol. 2 (2010),
<http://web.mit.edu/dimitrib/www/dpcchapter.html>
- Bertsekas, D.P.: Approximate policy iteration: A survey and some new methods. *Journal of Control Theory and Applications* 9(3), 310–335 (2011a)
- Bertsekas, D.P.: Temporal difference methods for general projected equations. *IEEE Transactions on Automatic Control* 56(9), 2128–2139 (2011b)

- Bertsekas, D.P., Ioffe, S.: Temporal differences-based policy iteration and applications in neuro-dynamic programming. Tech. Rep. LIDS-P-2349, Massachusetts Institute of Technology, Cambridge, US (1996).
<http://web.mit.edu/dimitrib/www/Tempdif.pdf>
- Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Athena Scientific (1996)
- Bertsekas, D.P., Borkar, V., Nedić, A.: Improved temporal difference methods with linear function approximation. In: Si, J., Barto, A., Powell, W. (eds.) Learning and Approximate Dynamic Programming. IEEE Press (2004)
- Boyan, J.: Technical update: Least-squares temporal difference learning. Machine Learning 49, 233–246 (2002)
- Bradtko, S.J., Barto, A.G.: Linear least-squares algorithms for temporal difference learning. Machine Learning 22(1-3), 33–57 (1996)
- Buşoniu, L., Babuška, R., De Schutter, B., Ernst, D.: Reinforcement Learning and Dynamic Programming Using Function Approximators. In: Automation and Control Engineering. Taylor & Francis, CRC Press (2010a)
- Buşoniu, L., De Schutter, B., Babuška, R., Ernst, D.: Using prior knowledge to accelerate online least-squares policy iteration. In: 2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR-2010), Cluj-Napoca, Romania (2010b)
- Buşoniu, L., Ernst, D., De Schutter, B., Babuška, R.: Approximate dynamic programming with a fuzzy parameterization. Automatica 46(5), 804–814 (2010c)
- Buşoniu, L., Ernst, D., De Schutter, B., Babuška, R.: Online least-squares policy iteration for reinforcement learning control. In: Proceedings 2010 American Control Conference (ACC-2010), Baltimore, US, pp. 486–491 (2010d)
- Dimitrakakis, C., Lagoudakis, M.: Rollout sampling approximate policy iteration. Machine Learning 72(3), 157–171 (2008)
- Engel, Y., Mannor, S., Meir, R.: Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In: Proceedings 20th International Conference on Machine Learning (ICML-2003), Washington, US, pp. 154–161 (2003)
- Engel, Y., Mannor, S., Meir, R.: Reinforcement learning with Gaussian processes. In: Proceedings 22nd International Conference on Machine Learning (ICML-2005), Bonn, Germany, pp. 201–208 (2005)
- Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research 6, 503–556 (2005)
- Farahmand, A.M., Ghavamzadeh, M., Szepesvári, C.S., Mannor, S.: Regularized policy iteration. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) Advances in Neural Information Processing Systems, vol. 21, pp. 441–448. MIT Press (2009)
- Geramifard, A., Bowling, M.H., Sutton, R.S.: Incremental least-squares temporal difference learning. In: Proceedings 21st National Conference on Artificial Intelligence and 18th Innovative Applications of Artificial Intelligence Conference (AAAI-2006), Boston, US, pp. 356–361 (2006)
- Geramifard, A., Bowling, M., Zinkevich, M., Sutton, R.S.: iLSTD: Eligibility traces & convergence analysis. In: Schölkopf, B., Platt, J., Hofmann, T. (eds.) Advances in Neural Information Processing Systems, vol. 19, pp. 440–448. MIT Press (2007)
- Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. Johns Hopkins (1996)
- Jung, T., Polani, D.: Kernelizing LSPE(λ). In: Proceedings 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-2007), Honolulu, US, pp. 338–345 (2007a)
- Jung, T., Polani, D.: Learning RoboCup-keepaway with kernels. In: Gaussian Processes in Practice, JMLR Workshop and Conference Proceedings, vol. 1, pp. 33–57 (2007b)

- Kolter, J.Z., Ng, A.: Regularization and feature selection in least-squares temporal difference learning. In: Proceedings 26th International Conference on Machine Learning (ICML-2009), Montreal, Canada, pp. 521–528 (2009)
- Konda, V.: Actor-critic algorithms. PhD thesis, Massachusetts Institute of Technology, Cambridge, US (2002)
- Lagoudakis, M., Parr, R., Littman, M.: Least-squares Methods in Reinforcement Learning for Control. In: Vlahavas, I.P., Spyropoulos, C.D. (eds.) SETN 2002. LNCS (LNAI), vol. 2308, pp. 249–260. Springer, Heidelberg (2002)
- Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. *Journal of Machine Learning Research* 4, 1107–1149 (2003a)
- Lagoudakis, M.G., Parr, R.: Reinforcement learning as classification: Leveraging modern classifiers. In: Proceedings 20th International Conference on Machine Learning (ICML-2003), Washington, US, pp. 424–431 (2003b)
- Lazaric, A., Ghavamzadeh, M., Munos, R.: Analysis of a classification-based policy iteration algorithm. In: Proceedings 27th International Conference on Machine Learning (ICML-2010), Haifa, Israel, pp. 607–614 (2010a)
- Lazaric, A., Ghavamzadeh, M., Munos, R.: Finite-sample analysis of LSTD. In: Proceedings 27th International Conference on Machine Learning (ICML-2010), Haifa, Israel, pp. 615–622 (2010b)
- Li, L., Littman, M.L., Mansley, C.R.: Online exploration in least-squares policy iteration. In: Proceedings 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2009), Budapest, Hungary, vol. 2, pp. 733–739 (2009)
- Maei, H.R., Szepesvári, C., Bhatnagar, S., Sutton, R.S.: Toward off-policy learning control with function approximation. In: Proceedings 27th International Conference on Machine Learning (ICML-2010), Haifa, Israel, pp. 719–726 (2010)
- Maillard, O.A., Munos, R., Lazaric, A., Ghavamzadeh, M.: Finite-sample analysis of Bellman residual minimization, vol. 13, pp. 299–314 (2010)
- Meyn, S., Tweedie, L.: Markov chains and stochastic stability. Springer, Heidelberg (1993)
- Moore, A.W., Atkeson, C.R.: The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning* 21(3), 199–233 (1995)
- Munos, R.: Error bounds for approximate policy iteration. In: Proceedings 20th International Conference (ICML-2003), Washington, US, pp. 560–567 (2003)
- Munos, R.: Approximate dynamic programming. In: *Markov Decision Processes in Artificial Intelligence*. Wiley (2010)
- Munos, R., Szepesvári, C.S.: Finite time bounds for fitted value iteration. *Journal of Machine Learning Research* 9, 815–857 (2008)
- Nedić, A., Bertsekas, D.P.: Least-squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems: Theory and Applications* 13(1-2), 79–110 (2003)
- Rasmussen, C.E., Kuss, M.: Gaussian processes in reinforcement learning. In: Thrun, S., Saul, L.K., Schölkopf, B. (eds.) *Advances in Neural Information Processing Systems*, vol. 16. MIT Press (2004)
- Scherrer, B.: Should one compute the Temporal Difference fix point or minimize the Bellman Residual? the unified oblique projection view. In: Proceedings 27th International Conference on Machine Learning (ICML-2010), Haifa, Israel, pp. 959–966 (2010)
- Schweitzer, P.J., Seidmann, A.: Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications* 110(2), 568–582 (1985)

- Sutton, R., Maei, H., Precup, D., Bhatnagar, S., Silver, D., Szepesvari, C.S., Wiewiora, E.: Fast gradient-descent methods for temporal-difference learning with linear function approximation. In: Proceedings 26th International Conference on Machine Learning (ICML-2009), Montreal, Canada, pp. 993–1000 (2009a)
- Sutton, R.S.: Learning to predict by the method of temporal differences. *Machine Learning* 3, 9–44 (1988)
- Sutton, R.S., Szepesvári, C.S., Maei, H.R.: A convergent $O(n)$ temporal-difference algorithm for off-policy learning with linear function approximation. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) *Advances in Neural Information Processing Systems*, vol. 21, pp. 1609–1616. MIT Press (2009b)
- Szepesvári, C.S.: *Algorithms for Reinforcement Learning*. Morgan & Claypool Publishers (2010)
- Taylor, G., Parr, R.: Kernelized value function approximation for reinforcement learning. In: Proceedings 26th International Conference on Machine Learning (ICML-2009), Montreal, Canada, pp. 1017–1024 (2009)
- Thiery, C., Scherrer, B.: Least-squares λ policy iteration: Bias-variance trade-off in control problems. In: Proceedings 27th International Conference on Machine Learning (ICML-2010), Haifa, Israel, pp. 1071–1078 (2010)
- Tsitsiklis, J.N.: On the convergence of optimistic policy iteration. *Journal of Machine Learning Research* 3, 59–72 (2002)
- Tsitsiklis, J.N., Van Roy, B.: An analysis of temporal difference learning with function approximation. *IEEE Transactions on Automatic Control* 42(5), 674–690 (1997)
- Xu, X., Xie, T., Hu, D., Lu, X.: Kernel least-squares temporal difference learning. *International Journal of Information Technology* 11(9), 54–63 (2005)
- Xu, X., Hu, D., Lu, X.: Kernel-based least-squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks* 18(4), 973–992 (2007)
- Yu, H.: Convergence of least squares temporal difference methods under general conditions. In: Proceedings 27th International Conference on Machine Learning (ICML-2010), Haifa, Israel, pp. 1207–1214 (2010)
- Yu, H., Bertsekas, D.P.: Convergence results for some temporal difference methods based on least squares. *IEEE Transactions on Automatic Control* 54(7), 1515–1531 (2009)
- Yu, H., Bertsekas, D.P.: Error bounds for approximations from projected linear equations. *Mathematics of Operations Research* 35(2), 306–329 (2010)

Chapter 4

Learning and Using Models

Todd Hester and Peter Stone

Abstract. As opposed to model-free RL methods, which learn directly from experience in the domain, model-based methods learn a model of the transition and reward functions of the domain on-line and plan a policy using this model. Once the method has learned an accurate model, it can plan an optimal policy on this model without any further experience in the world. Therefore, when model-based methods are able to learn a good model quickly, they frequently have improved sample efficiency over model-free methods, which must continue taking actions in the world for values to propagate back to previous states. Another advantage of model-based methods is that they can use their models to plan multi-step exploration trajectories. In particular, many methods drive the agent to explore where there is uncertainty in the model, so as to learn the model as fast as possible. In this chapter, we survey some of the types of models used in model-based methods and ways of learning them, as well as methods for planning on these models. In addition, we examine the typical architectures for combining model learning and planning, which vary depending on whether the designer wants the algorithm to run on-line, in batch mode, or in real-time. One of the main performance criteria for these algorithms is sample complexity, or how many actions the algorithm must take to learn. We examine the sample efficiency of a few methods, which are highly dependent on having intelligent exploration mechanisms. We survey some approaches to solving the exploration problem, including Bayesian methods that maintain a belief distribution over possible models to explicitly measure uncertainty in the model. We show some empirical comparisons of various model-based and model-free methods on two example domains before

Todd Hester
Department of Computer Science, The University of Texas at Austin, 1616 Guadalupe,
Suite 2.408, Austin, TX 78701
e-mail: todd@cs.utexas.edu

Peter Stone
Department of Computer Science, The University of Texas at Austin, 1616 Guadalupe,
Suite 2.408, Austin, TX 78701
e-mail: pstone@cs.utexas.edu

concluding with a survey of current research on scaling these methods up to larger domains with improved sample and computational complexity.

4.1 Introduction

The reinforcement learning (RL) methods described in the book thus far have been *model-free* methods, where the algorithm updates its value function directly from experience in the domain. *Model-based* methods (or *indirect* methods), however, perform their updates from a model of the domain, rather than from experience in the domain itself. Instead, the model is learned from experience in the domain, and then the value function is updated by planning over the learned model. This sequence is shown in Figure 4.1. This planning can take the form of simply running a model-free method on the model, or it can be a method such as *value iteration* or *Monte Carlo Tree Search*.

The models learned by these methods can vary widely. Models can be learned entirely from scratch, the structure of the model can be given so that only parameters need to be learned, or a nearly complete model can be provided. If the algorithm can learn an accurate model quickly enough, model-based reinforcement learning can be more sample efficient (take fewer actions to learn) than model-free methods. Once an accurate model is learned, an optimal policy can be planned without requiring any additional experiences in the world. For example, when an agent first discovers a goal state, the values of its policy can be updated at once through planning over its new model that represents that goal. Conversely, a model-free method would have to follow a trajectory to the goal many times for the values to propagate all the way back to the start state. This higher sample efficiency typically comes at the cost of more computation for learning the model and planning a policy and more space to represent the model.

Another advantage of models is that they provide an opportunity for the agent to perform *targeted* exploration. The agent can plan a policy using its model to drive

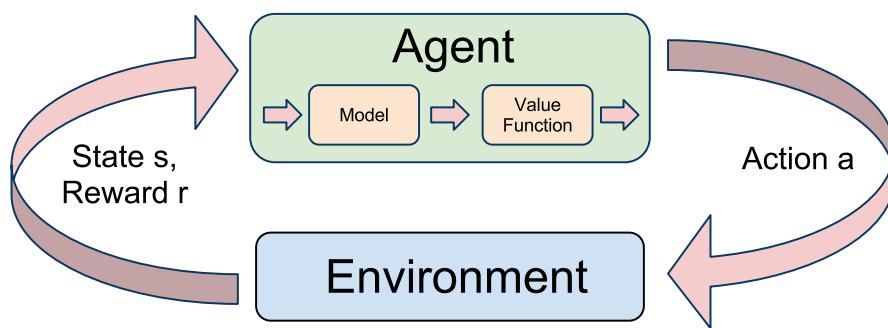


Fig. 4.1 Model-based RL agents use their experiences to first learn a model of the domain, and then use this model to compute their value function

the agent explore particular states; these states can be states it has not visited or is uncertain about. A key to learning a model quickly is acquiring the right experiences needed to learn the model (similar to *active learning*). Various methods for exploring in this way exist, leading to fast learning of accurate models, and thus high sample efficiency.

The main components of a model-based RL method are the model, which is described in Section 4.2, and the planner which plans a policy on the model, described in Section 4.3. Section 4.4 discusses how to combine models and planners into a complete agent. The sample complexity of the algorithm (explained in Section 4.5) is one of the main performance criteria that these methods are evaluated on. We examine algorithms for factored domains in Section 4.6. Exploration is one of the main focuses for improving sample complexity and is explained in Section 4.7. We discuss extensions to continuous domains in Section 4.8, examine the performance of some algorithms empirically in Section 4.9, and look at work on scaling up to larger and more realistic problems in Section 4.10. Finally, we conclude the chapter in Section 4.11.

4.2 What Is a Model?

A model is the information an agent would need in order to simulate the outcome of taking an action in the Markov Decision Process (MDP). If the agent takes action a from state s , the model should predict the next state, s' , and the reward, r . If the domain is stochastic, the model may provide a full probability distribution over next states, or it may be a *generative* model that simply provides a sample from the next state distribution. The model is updated as the agent interacts with its environment and is then used with a planning algorithm to calculate a policy.

A common approach to model-learning is to learn a tabular maximum likelihood model. In this case, the algorithm maintains a count, $C(s,a)$, of the times action a was taken from state s as well as a count, $C(s,a,s')$, of the number of times that each next state s' was reached from (s,a) . The probability of outcome s' is then:

$$P(s'|s,a) = T(s,a,s') = C(s,a,s')/C(s,a)$$

The algorithm also maintains the sum of rewards it has received from each transition, $Rsum(s,a)$, and computes the expected reward for a particular state-action to be the mean reward received from that state-action:

$$R(s,a) = Rsum(s,a)/C(s,a)$$

This tabular model learning is very straightforward and is used in some common model-based algorithms such as R-MAX (Brafman and Tennenholtz, 2001).

Table 4.1 shows an example of the counts and predictions of a tabular model after 50 experiences in the domain shown in Figure 4.2. Based on the empirical counts of each experienced transition, the model can make predictions about the actual transition probabilities, $T(s,a,s')$, and reward, $R(s,a)$. While the model's predictions for action 0 from state 1 are exactly correct, the probabilities for the other state-action pairs are only approximately correct, but will improve with more samples.

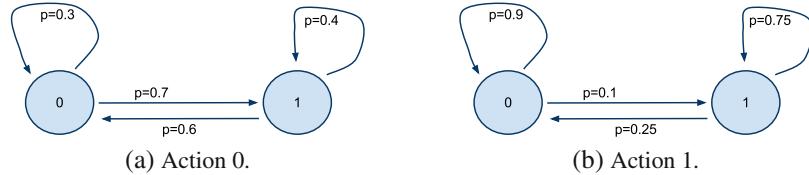


Fig. 4.2 This figure shows the transition probabilities for a simple two state MDP for both actions 0 and 1. We assume a reward of -1 for transitions leading to state 0, and a reward of +1 for transitions leading to state 1.

Table 4.1 This table shows an example of the model learned after 50 experiences in the domain shown in Figure 4.2

State	Action	C(s,a)	C(s,a,0)	C(s,a,1)	RSum(s,a)	T(s,a,0)	T(s,a,1)	R(s,a)
0	0	12	4	8	4	0.33	0.67	0.33
0	1	14	13	1	-12	0.93	0.07	-0.86
1	0	10	6	4	-2	0.6	0.4	-0.2
1	1	14	4	10	6	0.29	0.71	0.43

Learning a tabular model of the domain can be viewed as learning a separate model for every state-action in the domain, which can require the agent to take many actions to learn a model. If we assume that the transition dynamics may be similar across state-action pairs, we can improve upon tabular models by incorporating *generalization* into the model learning. Model learning can be viewed as a supervised learning problem with (s, a) as the input and s' and r as the outputs the supervised learner is trying to predict. One of the benefits of incorporating generalization is that the supervised learner will make predictions about the model for unseen states based on the transitions it has been trained on.

A number of approaches from the field of adaptive dynamic programming use neural networks to learn a model of the domain (Prokhorov and Wunsch, 1997; Venayagamoorthy et al, 2002). These approaches train the neural network to estimate the next state values based on previous state values. The neural network model is used within an actor-critic framework, with the predictions of the model used as inputs for the critic neural network that estimates the value of a given state.

Learning a model with a supervised learning technique inherently incorporates generalization into the model learning. With this approach, the supervised learner will make predictions about the transitions from unseen state-action pairs based on the state-action pairs it has been trained on. In many domains, it is easier to generalize the *relative* effects of transitions rather than their absolute outcomes. The relative transition effect, s^r , is the difference between the next state and the current state:

$$s^r = s' - s$$

For example, in a robotic control task, it may be easier to generalize that a given action increases the angle of a joint, rather than trying to generalize the absolute value of the joint angle after the action. This approach is taken in a number of

algorithms. Random forests are used to model the relative transition effects in (Hester and Stone, 2010). Jong and Stone (2007) make predictions for a given state-action pair based on the average of the relative effects of its nearest neighbors.

One of the earliest model-based RL methods used locally weighted regression (LWR) to learn models (Schaal and Atkeson, 1994; Atkeson et al, 1997). All experiences of the agent were saved in memory, and when a given state-action was queried, a locally weighted regression model was formed to provide a prediction of the average next state for the queried state-action pair. Combined with a unique exploration mechanism, this algorithm was able to learn to control a robot juggling.

4.3 Planning

Once the agent has learned an approximate model of the domain dynamics, the model can be used to learn an improved policy. Typically, the agent would re-plan a policy on the model each time it changes. Calculating a policy based on a model is called *planning*. These methods can also be used for planning a policy on a provided model, rather than planning while learning the model online. One option for planning on the model is to use the dynamic programming methods described in Chapter 1, such as Value Iteration or Policy Iteration. Another option is to use Monte Carlo methods, and particularly Monte Carlo Tree Search (MCTS) methods, described below. The main difference between the two classes of methods is that the dynamic programming methods compute the value function for the entire state space, while MCTS focuses its computation on the states that the agent is likely to encounter soon.

4.3.1 Monte Carlo Methods

Chapter 1 describes how to use Monte Carlo methods to compute a policy while interacting with the environment. These methods can be used in a similar way on experiences simulated using a learned model. The methods simulate a full trajectory of experience until the end of an episode or to a maximum search depth. Each simulated trajectory is called a *roll-out*. They then update the values of states along that trajectory towards the discounted sum of rewards received after visiting that state. These methods require only a generative model of the environment rather than a full distribution of next states. A variety of MCTS methods exist which vary in how they choose actions at each state in their search.

Monte Carlo Tree Search methods build a tree of visited state-action pairs out from the current state, shown in Figure 4.3. This tree enables the algorithm to reuse information at previously visited states. In vanilla MCTS, the algorithm takes greedy actions to a specified depth in the tree, and then takes random actions from there until the episode terminates. This tree search focuses value updates on the states that MCTS visits between the agent’s current state and the end of the roll-out,

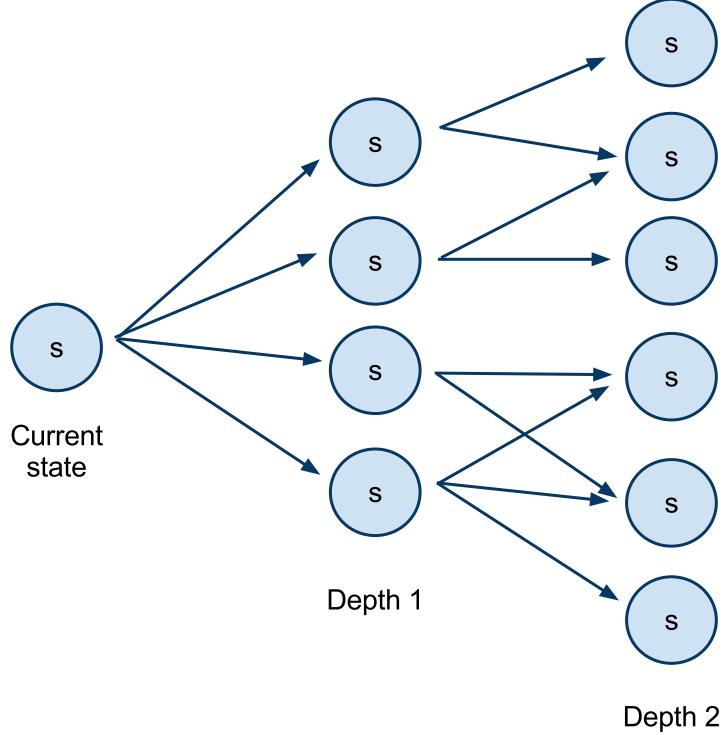


Fig. 4.3 This figure shows a Monte Carlo Tree Search roll-out from the agent's current state out to a depth of 2. The MCTS methods simulate a trajectory forward from the agent's current state. At each state, they select some action, leading them to a next state one level deeper in the tree. After rolling out to a terminal state or a maximum depth, the value of the actions selected are updated towards the rewards received following it on that trajectory.

which can be more efficient than planning over the entire state space as dynamic programming methods do. We will discuss a few variants of MCTS that vary in how they select actions at each state to efficiently find a good policy.

Sparse sampling (Kearns et al, 1999) was a pre-cursor to the MCTS planning methods discussed in this chapter. The authors determine the number, C , of samples of next states required to accurately estimate the value of a given state-action, (s,a) , based on the maximum reward in the domain, R_{max} , and the discount factor, γ . They also determine the horizon h that must be searched to given the discount factor γ . The estimate of the value of a state-action pair at depth t is based on C samples of the next states at depth $t + 1$. The value of each of these next states is based on C samples of each of the actions at that state, and so on up to horizon h . Instead of

sampling trajectories one at a time like MCTS does, this method expands the tree one level deeper each step until reaching the calculated horizon. The algorithm's running time is $O((|A||C|^h)$. As a sampling-based planning method that is proven to converge to accurate values, sparse sampling provides an important theoretical basis for the MCTS methods that follow.

UCT (Kocsis and Szepesvári, 2006) is another MCTS variant that improves upon the running time of Sparse Sampling by focusing its samples on the most promising actions. UCT searches from the start state to the end, selecting actions based on upper confidence bounds using the UCB1 algorithm (Auer et al, 2002). The algorithm maintains a count, $C(s,d)$, of visits to each state at a given depth in the search, d , as well as a count, $C(s,a,d)$, of the number of times action a was taken from that state at that depth. These counts are used to calculate the upper confidence bound to select the action. The action selected at each step is calculated with the following equation (where C_p is an appropriate constant based on the range of rewards in the domain):

$$a = \operatorname{argmax}_a Q^d(s,a) + 2C_p \sqrt{\frac{\log(C(s,d))}{C(s,a,d)}}$$

By selecting actions using the upper tail of the confidence interval, the algorithm mainly samples good actions, while still exploring when other actions have a higher upper confidence bound. Algorithm 10 shows pseudo-code for the UCT algorithm, which is run from the agent's current state, s , with a depth, d , of 0. The algorithm is provided with a learning rate, α , and the range of one-step rewards in the domain, r_{range} . Line 6 of the algorithm recursively calls the UCT method, to sample an action at the next state one level deeper in the search tree. Modified versions of UCT have had great success in the world of Go algorithms as a planner with the model of the game already provided (Wang and Gelly, 2007). UCT is also used as the planner inside several model-based reinforcement learning algorithms (Silver et al, 2008; Hester and Stone, 2010).

```

1: if TERMINAL or d == MAXDEPTH then
2:   return 0
3:    $a \leftarrow \operatorname{argmax}_{a'}(Q^d(s,a') + 2 \cdot r_{range} \cdot \sqrt{\log(c(s,d))/c(s,a',d)})$ 
4:    $(s',r) \leftarrow \text{SAMPLENEXTSTATE}(s,a)$ 
5:    $retval \leftarrow r + \text{UCT}(s',d+1,\alpha)$ 
6:    $c(s,d) \leftarrow c(s,d) + 1$ 
7:    $c(s,a,d) \leftarrow c(s,a,d) + 1$ 
8:    $Q^d(s,a') \leftarrow \alpha \cdot retval + (1 - \alpha) \cdot Q(s,a',d)$ 
9: return retval

```

Algorithm 10. UCT (Inputs s , d , α , r_{range})

Having separately introduced the ideas of model learning (Section 4.2) and planning (Section 4.3), we now discuss the challenges that arise when combining these two together into a full model-based method, and how these challenges have been addressed.

4.4 Combining Models and Planning

There are a number of ways to combine model learning and planning. Typically, as the agent interacts with the environment, its model gets updated at every time step with the latest transition, $\langle s, a, r, s' \rangle$. Each time the model is updated, the algorithm re-plans on it with its planner (as shown in Figure 4.4). This approach is taken by many algorithms (Brafman and Tennenholtz, 2001; Hester and Stone, 2009; Degris et al, 2006). However, due to the computational complexity of learning the model and planning on it, it is not always feasible.

Another approach is to do model updates and planning in batch mode, only processing them after every k actions, an approach taken in (Deisenroth and Rasmussen, 2011). However, this approach means that the agent takes long pauses between some actions while performing batch updates, which may not be acceptable in some problems.

DYNA (Sutton, 1990, 1991) is a reactive RL architecture. In it, the agent starts with either a real action in the world or a saved experience. Unlike value iteration, where Bellman updates are performed on all the states by iterating over the state space, here planning updates are performed on randomly selected state-action pairs. The algorithm updates the action-values for the randomly selected state-action using the Bellman equations, thus updating its policy. In this way, the real actions require only a single action-value update, while many model-based updates can take place in between the real actions. While the DYNA framework separates the model updates from the real action loop, it still requires many model-based updates for the policy to become optimal with respect to its model.

Prioritized Sweeping (Moore and Atkeson, 1993) (described in Chapter 1) improves upon the DYNA idea by selecting which state-action pairs to update based on priority, rather than selecting them randomly. It updates the state-action pairs in order, based on the expected change in their value. Instead of iterating over the entire state space, prioritized sweeping updates values propagating backwards across the state space from where the model changed.

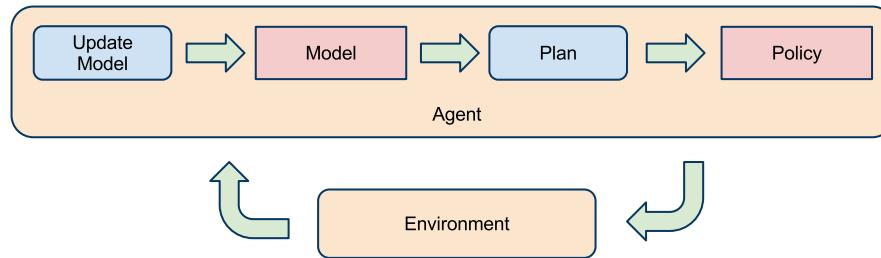


Fig. 4.4 Typically, model-based agent's interleave model learning and planning sequentially, first completing an update to the model, and then planning on the updated model to compute a policy

Using MCTS to plan instead of dynamic programming techniques such as Value Iteration or Policy Iteration leads to a slightly different architecture. With the dynamic programming techniques, a value function is computed for the entire state space after the model has changed. Once it is computed, dynamic programming does not need to be performed again unless the model changes. With sample-based MCTS methods that focus computation on the states the agent is likely to visit soon, planning roll-outs must occur every step to look ahead at likely future states. These methods are typically anytime algorithms, so each step can be limited to a given amount of computation if desired.

DYNA-2 is an architecture that extends the DYNA idea of updating its value function using both real and simulated experiences to using sample-based planning (Silver et al, 2008). DYNA-2 maintains separate linear function approximators to represent both *permanent* and *transient* memories of the value function. The permanent memory is updated through real experiences in the world. Between each action in the world, the transient memory is updated by running UCT on the agent's model of the world. The transient memory is focused on a narrower part of the state space (where UCT is sampling right now) and is used to augment the global value function. Actions are selected based on the combination of the permanent and transient value functions. This architecture combines a rough permanent global value function with a more refined transient local value function created by UCT planning each step.

For some problems, such as controlling robots or other physical devices, it may desirable to have a *real-time* architecture. In the previous architectures that we have described, it is possible for the model update or planning steps to take a significant amount of time. Hester et al (2011) developed a threaded architecture for real-time model-based RL that puts the model learning, planning, and acting in three parallel threads, shown in Figure 4.5. The threads communicate through four shared data structures: a list of experiences to be added to the model, a copy of the model that the planner uses, the current state for the planner to plan from, and the agent's policy. The model-learning thread runs in a loop, removing experiences from the list of new experiences, updating its model with these experiences, and then copying the model over to a version used by the planner. The planning thread runs a sample-based planning algorithm (such as UCT) on its version of the model, planning from the agent's current state, and updating the policy. The action thread adds the latest experience to the update list, sets the agent's current state for planning, and returns the best action for that state from its policy. The action thread is able to return an action immediately at whatever frequency is required. Depending on the action frequency and the length of time it takes to update the model, the model learning thread can be waiting for new experiences each time, or updating many new experiences into the model at a time. When the model is incorporating many experiences at a time, the algorithm's behavior is similar to batch learning methods, however the agent will continue taking actions while the batch updates takes place¹.

¹ Source code for this architecture is available at:
http://www.ros.org/wiki/rl_agent

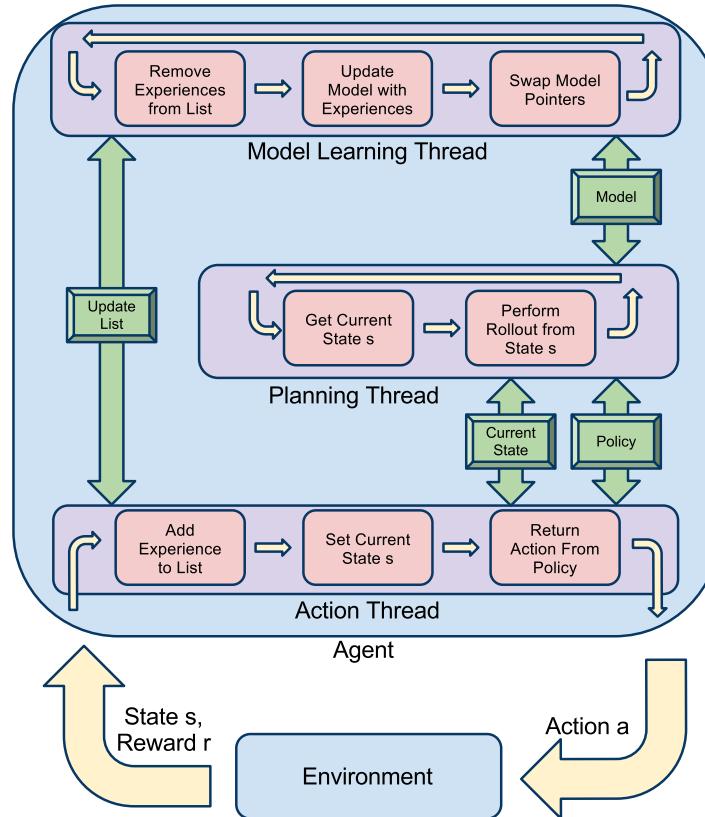


Fig. 4.5 The parallel architecture for real-time model-based RL proposed by Hester et al (2011). There are three separate parallel threads for model learning, planning, and acting. Separating model learning and planning from the action selection enables it to occur at the desired rate regardless of the time taken for model learning or planning.

4.5 Sample Complexity

A key benefit of model-based methods in comparison to model-free ones is that they can provide better *sample efficiency* than model-free methods. The *sample complexity* of an algorithm refers to the number of actions that an algorithm takes to learn an optimal policy. The sample complexity of *exploration* is the number of sub-optimal exploratory actions the agent must take. Kakade (2003) proves the lower bound for this sample complexity is $O(\frac{NA}{\varepsilon(1-\gamma)} \log \frac{1}{\delta})$ for stochastic domains, where N is the number of states, A is the number of actions, γ is the discount factor, and the algorithm finds an ε -optimal policy (a policy whose value is within ε of optimal) with probability $1 - \delta$.

Model-based methods can be more sample efficient than model-free approaches because they can update their value functions from their model rather than having to take actions in the real world. For this to be effective, however, the method must have a reasonable model of the world to plan on. Therefore, the main restriction on the sample efficiency of these methods is the number of actions it takes them to learn an accurate model of the domain. The methods presented in this section drive the agent to acquire the samples necessary to learn an accurate model quickly.

The Explicit Explore or Exploit (E^3) algorithm (Kearns and Singh, 1998) was the first algorithm to be proved to converge to a near-optimal policy in polynomial time². The authors analyze their algorithm in terms of a mixing time, T , which is the horizon over which the value of the policy is calculated. For discounted domains,

$$T = \frac{1}{1 - \gamma}$$

The algorithm maintains counts of visits to each state and considers states with fewer than m visits to be unknown. When the agent reaches an unknown state, it performs balanced wandering (taking the action it has selected the least from that state). If it reaches a known state, it attempts to plan an exploration policy that gets to an unknown state as quickly as possible. If the probability of reaching an unknown state is greater than $\varepsilon/(2R_{max})$, then this policy is followed. If that bound is not exceeded, then the authors have proven that planning an optimal policy *must* result in a policy that is within ε of optimal. Therefore, if the probability of reaching an unknown state does not reach that bound, the algorithm plans an approximate optimal policy and follows it. With probability no less than $1 - \delta$, the E^3 algorithm will attain a return greater than the optimal value $-\varepsilon$, in a number of steps polynomial in N , T , R_{max} , $\frac{1}{\varepsilon}$, and $\frac{1}{\delta}$.

R-MAX (Brafman and Tennenholtz, 2001) is a similar model-based algorithm with proven bounds on its sample efficiency. It also learns a tabular maximum likelihood model and tracks known and unknown states. The key insight of the R-MAX algorithm is 'optimism in the face of uncertainty'. The algorithm replaces unknown transitions in its model with transitions to an *absorbing state*. An absorbing state is a state where all actions leave the agent in the absorbing state and provide the agent with the maximum reward in the domain, R_{max} . Doing so encourages the agent to explore all state-action pairs m times, allowing it to learn an accurate model and thus an optimal policy. This algorithm is simpler than the E^3 algorithm, and it employs an *implicit* explore or exploit approach rather than explicitly deciding between two policies as E^3 does. With probability no less than $1 - \delta$, the R-MAX algorithm will attain an expected return with 2ε of optimal within a number of steps polynomial in $N, A, T, \frac{1}{\varepsilon}$, and $\frac{1}{\delta}$. Pseudo-code for the R-MAX algorithm is shown in Algorithm 11³.

The principles used in R-MAX are formalized in a learning framework called Knows What It Knows (KWIK) (Li et al, 2008). For a model learning method to fit

² Q-LEARNING, which was developed before E^3 , was not proved to converge in polynomial time until after the development of E^3 (Even-dar and Mansour, 2001).

³ Source code for R-MAX is available at: http://www.ros.org/wiki/r1_agent

```

1: // Initialize  $s_r$  as absorbing state with reward  $R_{max}$ 
2: for all  $a \in A$  do
3:    $R(s_r, a) \leftarrow R_{max}$ 
4:    $T(s_r, a, s_r) \leftarrow 1$ 
5: Initialize  $s$  to a starting state in the MDP
6: loop
7:   Choose  $a = \pi(s)$ 
8:   Take action  $a$ , observe  $r, s'$ 
9:   // Update model
10:  Increment  $C(s, a, s')$ 
11:  Increment  $C(s, a)$ 
12:   $RSUM(s, a) \leftarrow RSUM(s, a) + r$ 
13:  if  $C(s, a) \geq m$  then
14:    // Known state, update model using experience counts
15:     $R(s, a) \leftarrow RSUM(s, a) / C(s, a)$ 
16:    for all  $s' \in C(s, a, \cdot)$  do
17:       $T(s, a, s') \leftarrow C(s, a, s') / C(s, a)$ 
18:  else
19:    // Unknown state, set optimistic model transition to absorbing state
20:     $R(s, a) \leftarrow R_{max}$ 
21:     $T(s, a, s_r) \leftarrow 1$ 
22:  // Plan policy on updated model
23:  Call VALUE-ITERATION
24:   $s \leftarrow s'$ 

```

Algorithm 11. R-Max (Inputs S, A, m, R_{max})

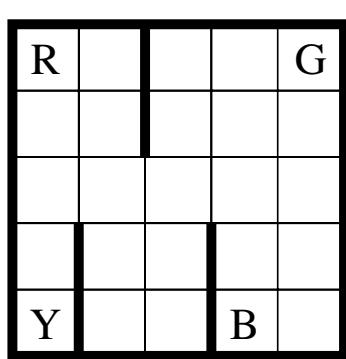
the KWIK framework, when queried about a particular state-action pair, it must always either make an accurate prediction, or reply “I don’t know” and request a label for that example. KWIK algorithms can be used as the model learning methods in an RL setting, as the agent can be driven to explore the states where the model replies “I don’t know” to improve its model quickly. The drawback of KWIK algorithms is that they often require a large number of experiences to guarantee an accurate prediction when not saying “I don’t know.”

4.6 Factored Domains

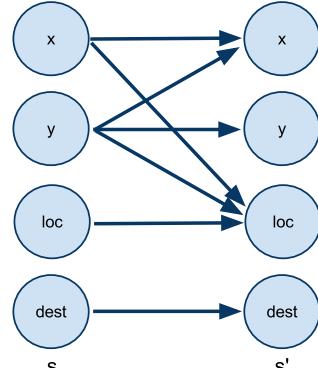
Many problems utilize a *factored* state representation where the state is represented by a vector of n state features:

$$s = \langle x_0, \dots, x_n \rangle$$

For example, an agent learning to control a robot could represent the state with a separate state feature for each joint. In many cases, the transition probabilities in factored domains are assumed to be determined by a Dynamic Bayesian Network (DBN). In the DBN model, each state feature of the next state may only be



(a) Taxi Domain.



(b) DBN Transition Structure.

Fig. 4.6 4.6a shows the Taxi domain. 4.6b shows the DBN transition model for this domain. Here the x feature in state s' is only dependent on the x and y features in state s and the y feature is only dependent on the previous y . The passenger's *destination* is only dependent on its previous *destination*, and her current *location* is dependent on her location the step before as well as the taxi's x,y location the step before.

dependent on some subset of features from the previous state and action. The features that a given state feature are dependent on are called its *parents*. The maximum number of parents that any of the state features has is called the *maximum in-degree* of the DBN. When using a DBN transition model, it is assumed that each feature transitions independently of the others. These separate transition probabilities can be combined together into a prediction of the entire state transition with the following equation:

$$P(s'|s,a) = T(s,a,s') = \prod_{i=0}^n P(x_i|s,a)$$

Learning the structure of this DBN transition model is known as the *structure learning problem*. Once the structure of the DBN is learned, the *conditional probabilities* for each edge must be learned. Typically these probabilities are stored in a conditional probability table, or CPT.

Figure 4.6 shows an example DBN for the Taxi domain (Dietterich, 1998). Here the agent's state is made up of four features: its x and y location, the passenger's *location*, and the passenger's *destination*. The agent's goal is to navigate the taxi to the passenger, pick up the passenger, navigate to her destination, and drop off the passenger. The y location of the taxi is only dependent on its previous y location, and not its x location or the current location or destination of the passenger. Because of the vertical walls in the domain, the x location of the taxi is dependent on both x and y . If this structure is known, it makes the model learning problem much easier, as the same model for the transition of the x and y variables can be used for any possible value of the passenger's location and destination.

Model-based RL methods for factored domains vary in the amount of information and assumptions given to the agent. Most assume a DBN transition model,

and that the state features do transition independently. Some methods start with no knowledge of the model and must first learn the structure of the DBN and then learn the probabilities. Other methods are given the structure and must simply learn the probabilities associated with each edge in the DBN. We discuss a few of these variations below.

The DBN- E^3 algorithm (Kearns and Koller, 1999) extends the E^3 algorithm to factored domains where the structure of the DBN model is known. With the structure of the DBN already given, the algorithm must learn the probabilities associated with each edge in the DBN. The algorithm is able to learn a near-optimal policy in a number of actions polynomial in the number of parameters of the DBN-MDP, which can be exponentially smaller than the number of total states.

Similar to the extension of E^3 to DBN- E^3 , R-MAX can be extended to FACTORED-R-MAX for factored domains where the structure of the DBN transition model is given (Guestrin et al, 2002). This method achieves the same sample complexity bounds as the DBN- E^3 algorithm, while also maintaining the implementation and simplicity advantages of R-MAX over E^3 .

Structure Learning Factored R-MAX (SLF-R-MAX) (Strehl et al, 2007) applies an R-MAX type approach to factored domains where the structure of the DBN is not known. It learns the structure of the DBN as well as the conditional probabilities when given the maximum in-degree of the DBN. The algorithm enumerates all possible combinations of input features as elements and then creates counters to measure which elements are relevant. The algorithm makes predictions when a relevant element is found for a queried state; if none is found, the state is considered unknown. Similar to R-MAX, the algorithm gives a bonus of R_{max} to unknown states in value iteration to encourage the agent to explore them. The sample complexity of the algorithm is highly dependent on the maximum in-degree, D , of the DBN. With probability at least $1 - \delta$, the SLF-R-MAX algorithm's policy is ϵ -optimal except for at most k time steps, where:

$$k = O\left(\frac{n^{3+2D}AD\ln(\frac{nA}{\delta})\ln(\frac{1}{\delta})\ln(\frac{1}{\epsilon(1-\gamma)})}{\epsilon^3(1-\gamma)^6}\right)$$

Here, n is the number of factors in the domain, D is the maximum in-degree in the DBN, and γ is the discount factor.

Diuk et al (2009) improve upon the sample complexity of SLF-R-MAX with the k -Meteorologists R-MAX (MET-R-MAX) algorithm by introducing a more efficient algorithm for determining which input features are relevant for its predictions. It achieves this improved efficiency by using the mean squared error of the predictors based on different DBNs. This improves the sample complexity for discovering the structure of the DBN from $O(n^{2D})$ to $O(n^D)$. The overall sample complexity bound for this algorithm is the best known bound for factored domains.

Chakraborty and Stone (2011) present a similar approach that does not require knowledge of the in-degree of the DBN called Learn Structure and Exploit with R-MAX (LSE-R-MAX). It takes an alternative route to solving the structure learning problem in comparison to MET-R-MAX by assuming knowledge of a planning

horizon that satisfies certain conditions, rather than knowledge of the in-degree. With this assumption, it solves the structure learning problem in sample complexity bounds which are competitive with MET-R-MAX, and it performs better empirically in two test domains.

Decision trees present another approach to the structure learning problem. They are able to naturally learn the structure of the problem by using information gain to determine which features are useful to split on to make predictions. In addition, they can generalize more than strict DBN models can. Even for state features that are parents of a given feature, the decision tree can decide that in portions of the state space, that feature is not relevant. For example, in taxi, the location of the passenger is only dependent on the location of the taxi when the *pickup* action is being performed. In all other cases, the location of the passenger can be predicted while ignoring the taxi's current location.

Decision trees are used to learn models in factored domains in the SPITI algorithm (Degris et al, 2006). The algorithm learns a decision tree to predict each state feature in the domain. It plans on this model using Structured Value Iteration (Boutilier et al, 2000) and uses ε -greedy exploration. The generalization in its model gives it better sample efficiency than many methods using tabular or DBN models in practice. However, there are no guarantees that the decision tree will fully learn the correct transition model, and therefore no theoretical bounds have been proven for its sample efficiency.

RL-DT is another approach using decision trees in its model that attempts to improve upon SPITI by modeling the relative transitions of states and using a different exploration policy (Hester and Stone, 2009). By predicting the relative change in each feature, rather than its absolute value, the tree models are able to make better predictions about the transition dynamics for unseen states. In addition, the algorithm uses a more directed exploration scheme, following R-MAX type exploration of driving the agent to states with few visits until the agent finds a state with reward near R_{max} , at which point it switches to exploiting the policy computed using its model. This algorithm has been shown to be effective on gridworld tasks such as Taxi, as well as on humanoid robots learning to score penalty kicks (Hester et al, 2010).

Figure 4.7 shows an example decision tree predicting the relative change in the x variable of the agent in the given gridworld domain. The decision tree is able to split on both the actions and the state of the agent, allowing it to split the state space up into regions where the transition dynamics are the same. Each leaf of the tree can make probabilistic predictions based on the ratio of experienced outcomes in that leaf. The grid is colored to match the leaves on the left side of the tree, making predictions for when the agent takes the *east* action. The tree is built on-line while the agent is acting in the MDP. At the start, the tree will be empty, and will slowly be refined over time. The tree will make predictions about broad parts of the state space at first, such as what the *EAST* or *WEST* actions do, and eventually refine itself to have leaves for individual states where the transition dynamics differ from the global dynamics.

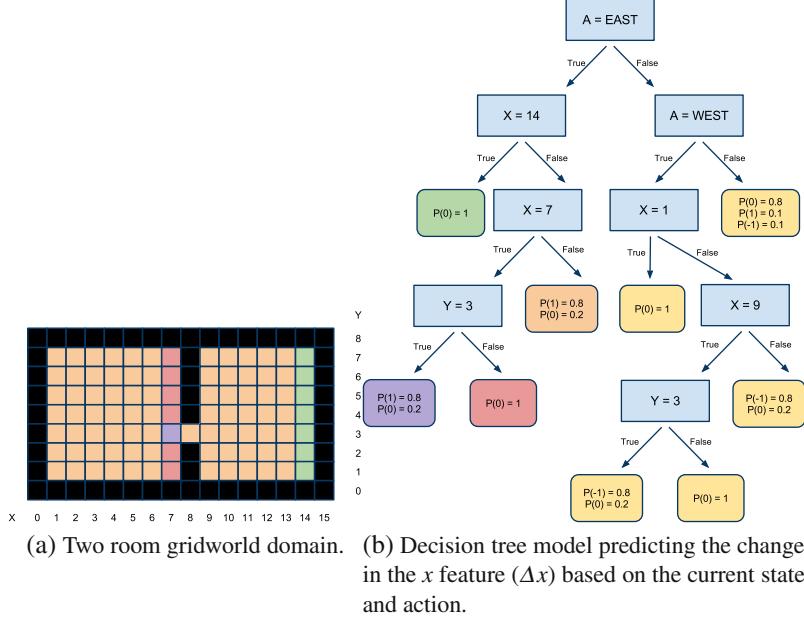


Fig. 4.7 This figure shows the decision tree model learned to predict the change in the x feature (or Δx). The two room gridworld is colored to match the corresponding leaves of the left side of the tree where the agent has taken the *east* action. Each rectangle represents a split in the tree and each rounded rectangle represents a leaf of the tree, showing the probabilities of a given value for Δx . For example, if the action is *east* and $x = 14$ we fall into the green leaf on the left, where the probability of Δx being 0 is 1.

4.7 Exploration

A key component of the E^3 and R-MAX algorithms is how and when the agent decides to take an exploratory (sub-optimal) action rather than exploit what it knows in its model. One of the advantages of model-based methods is that they allow the agent to perform *directed* exploration, planning out multi-step exploration policies rather than the simple ϵ -greedy or softmax exploration utilized by many model-free methods. Both the E^3 and R-MAX algorithms do so by tracking the number of visits to each state and driving the agent to explore all states with fewer than a given number of visits. However, if the agent can measure uncertainty in its model, it can drive exploration without depending on visit counts. The methods presented in this section follow this approach. They mainly vary in two dimensions: 1) how they measure uncertainty in their model and 2) exactly how they use the uncertainty to drive exploration.

Model-based Bayesian RL methods (Chapter 11) seek to solve the exploration problem by maintaining a posterior distribution over possible models. This approach is promising for solving the exploration problem because it provides a principled way to track the agent’s uncertainty in different parts of the model. In addition, with this explicit uncertainty measure, Bayesian methods can plan to explore states that have the potential to provide future rewards, rather than simply exploring states to reduce uncertainty for its own sake.

Duff (2003) presents an ‘optimal probe’ that solves the exploration problem optimally, using an augmented statespace that includes both the agent’s state in the world and its beliefs over its models (called a *belief state MDP*). The agent’s model includes both how an action will affect its state in the world, and how it will affect the agent’s beliefs over its models (and what model it will believe is most likely). By planning over this larger augmented state space, the agent can explore optimally. It knows which actions will change its model beliefs in significant and potentially useful ways, and can ignore actions that only affect parts of the model that will not be useful. While this method is quite sample efficient, planning over this augmented state space can be very computationally expensive. Wang et al (2005) make this method more computationally feasible by combining it with MCTS-like planning. This can be much more efficient than planning over the entire state space, as entire parts of the belief space can be ignored after a few actions. BEETLE (Poupart et al, 2006) takes a different approach to making this solution more computationally feasible by parametrizing the model and tying model parameters together to reduce the size of the model learning problem. However, this method is still impractical for any problem with more than a handful of states.

Other Bayesian methods use the model distribution to drive exploration without having to plan over a state space that is augmented with model beliefs. Both Bayesian DP (Strens, 2000) and Best of Sampled Set (BOSS) (Asmuth et al, 2009) approach the exploration problem by sampling from the distribution over world models and using these samples in different ways.

Bayesian DP samples a single model from the distribution, plans a policy using it, and follows that policy for a number of steps before sampling a new model. In between sampling new models, the agent will follow a policy consistent with the sampled model, which may be more exploratory or exploitative depending on the sampled model.

BOSS on the other hand, samples k models from the model posterior distribution whenever it collects sufficient data for some part of the model. It then merges the models into a single optimistic model with the same state space, but an augmented action space of kA actions. Essentially, there is an action modeled by each of the predictions of the k models for each of the A actions. Planning over this optimistic model allows the agent to select at each state an action from any of the k sampled models. The agent is driven to explore areas of the state space where the model is uncertain because due to the variance in the model distribution in that part of the state space, at least one of the sampled models is likely to be optimistic.

Model Based Bayesian Exploration (MBBE) is a similar approach taken by Dearden et al (1999). They maintain a distribution over model parameters and sample and solve k models to get a distribution over action-values. They use this distribution over action-values to calculate the value of perfect information (VPI) that is based on the uncertainty in the action-values along with the expected difference in reward that might be gained. They use the VPI value to give a bonus value to actions with high information gain.

These three methods (Bayesian DP, BOSS, and MBBE) provide three different approaches to sampling from a Bayesian distribution over models to solve the exploration problem. While these methods provide efficient exploration, they do require the agent to maintain Bayesian distributions over models and sample models from the distribution. They also require the user to create a well-defined model prior.

Kolter and Ng (2009) attempt to extend the polynomial convergence proofs of E^3 and R-MAX to Bayesian methods with an algorithm called Bayesian Exploration Bonus (BEB). BEB follows the Bayesian optimal policy after a polynomial number of steps, using exploration rewards similar to R-MAX to drive the agent towards the Bayesian optimal policy. One drawback of BEB is that it requires the transition functions to be drawn from a Dirichlet distribution.

Model Based Interval Estimation (MBIE) (Wiering and Schmidhuber, 1998; Strehl and Littman, 2005) is a similar approach that looks at the distribution over transition probabilities rather than action-value distributions. The algorithm maintains statistical confidence intervals over the transition probabilities where transitions that have been sampled more often have tighter distributions around the same mean. When selecting actions, the algorithm computes the value function according to the transitions probabilities that are both within the calculated confidence interval *and* result in the highest policy values. Effectively, MBIE solves for the maximum over likely transition probabilities in addition to the maximum over individual actions.

SLF-R-MAX, MET-R-MAX, and LSE-R-MAX perform directed exploration on factored domains (Strehl et al, 2007; Diuk et al, 2009; Chakraborty and Stone, 2011). They use an R-MAX type exploration bonus to explore to determine the structure of the DBN transition model and to determine the conditional probabilities. They can explore less than methods such as R-MAX since their DBN model should be able to determine that some features are not relevant for the predictions of certain features. With fewer relevant features, the space of states where the relevant features differ is much smaller than when assuming each state can have different transition dynamics.

With tabular models, it is clear that the agent must explore each state-action in order to learn an accurate model for each one. This task can be accomplished through random exploration, but it is more effective to direct exploration to the unvisited states like R-MAX does. When dealing with models that generalize such as decision trees or DBNs, however, we do not want the agent exploring every state. One of the benefits of these models is that they are able to make reasonable predictions about unseen state-action pairs. Therefore, instead of exploring every state-action pair, it

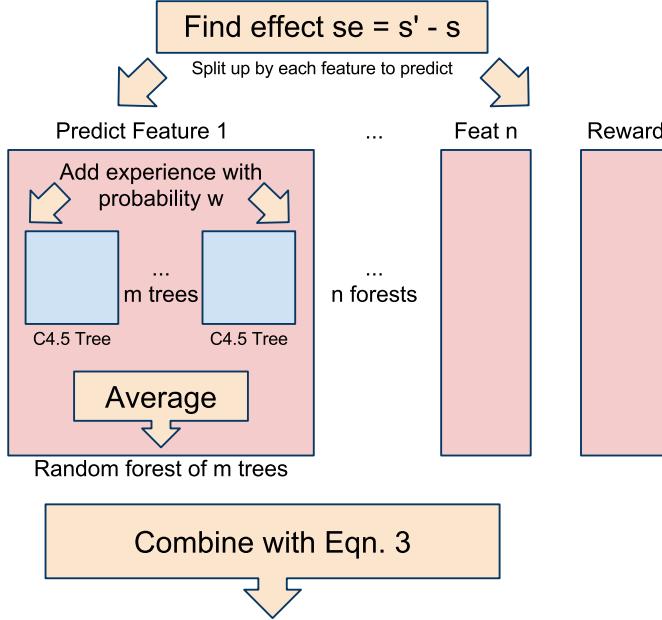


Fig. 4.8 *TEXPLOR’s Model Learning*. This figure shows how the TEXPLOR algorithm (Hester and Stone, 2010) learns a model of the domain. The agent calculates the difference between s' and s as the transition effect. Then it splits up the state vector and learns a random forest to predict the change in each state feature. Each random forest is made up of stochastic decision trees, which are updated with each new experience with probability w . The random forest’s predictions are made by averaging each tree’s predictions, and then the predictions for each feature are combined into a complete model of the domain.

would be more efficient for the agent to use an approach similar to the Bayesian methods such as BOSS described above.

The TEXPLOR algorithm (Hester and Stone, 2010) is a model-based method for factored domains that attempts to accomplish this goal. It learns multiple possible decision tree models of the domain, in the form of a random forest (Breiman, 2001). Figure 4.8 shows the random forest model. Each random forest model is made up of m possible decision tree models of the domain and trained on a subset of the agent’s experiences. The agent plans on the average of these m models and can include an exploration bonus based on the variance of the models’ predictions.

The TEXPLOR agent builds k hypotheses of the transition model of the domain, similar to the multiple sampled models of BOSS or MBBE. TEXPLOR combines the models differently, however, creating a merged model that predicts next state distributions that are the average of the distributions predicted by each of the models. Planning over this average distribution allows the agent to explore promising transitions when their probability is high enough (when many of the models predict the promising outcome, or one of them predicts it with high probability). This is similar to the approach of BOSS, but takes into account the number of sampled models that

predict the optimistic outcome. In TEXPLORE, the prediction of a given model has probability $\frac{1}{k}$, while the extra actions BOSS creates are always assumed to transition as that particular model predicts. This difference becomes clear with an example. If TEXPLORE's models disagree and the average model predicts there is a small chance of a particular negative outcome occurring, the agent will avoid it based on the chance that it may occur. The BOSS agent, however, will simply select an action from a different model and ignore the possibility of these negative rewards. On the other hand, if TEXPLORE's average model predicts a possibility of a high-valued outcome occurring, it may be worth exploring if the value of the outcome is high enough relative to its probability. TEXPLORE has been used in a gridworld with over 300,000 states (Hester and Stone, 2010) and run in real-time on an autonomous vehicle (Hester et al, 2011)⁴.

Schmidhuber (1991) tries to drive the agent to where the model has been improving the most, rather than trying to estimate where the model is poorest. The author takes a traditional model-based RL method, and adds a confidence module, which is trained to predict the absolute value of the error of the model. This module could be used to create intrinsic rewards encouraging the agent to explore high-error state-action pairs, but then the agent would be attracted to noisy states in addition to poorly-modeled ones. Instead the author adds another module that is trained to predict the changes in the confidence module outputs. Using this module, the agent is driven to explore the parts of the state space that most improve the model's prediction error.

Baranes and Oudeyer (2009) present an algorithm based on a similar idea called Robust Intelligent Adaptive Curiosity (R-IAC), a method for providing intrinsic reward to encourage a developing agent to explore. Their approach does not adopt the RL framework, but is similar in many respects. In it, they split the state space into regions and learn a model of the transition dynamics in each region. They maintain an estimate of the prediction error for each region and use the gradient of this error as the intrinsic reward for the agent, driving the agent to explore the areas where the prediction errors are improving the most. Since this approach is not using the RL framework, their algorithm selects actions only to maximize the immediate reward, rather than the discounted sum of future rewards. Their method has no way of incorporating external rewards, but it could be used to provide intrinsic rewards to an existing RL agent.

4.8 Continuous Domains

Most of the algorithms described to this point in the chapter assume that the agent operates in a discrete state space. However, many real-world problems such as robotic control involve continuously valued states and actions. These approaches can be extended to continuous problems by quantizing the state space, but very

⁴ Source code for the TEXPLORE algorithm is available at:
http://www.ros.org/wiki/rl_agent

fine discretizations result in a very large number of states, and some information is lost in the discretization. Model-free approaches can be extended fairly easily to continuous domains through the use of function approximation. However, there are multiple challenges that must be addressed to extend model-based methods to continuous domains: 1) learning continuous models, 2) planning in a continuous state space, and 3) exploring a continuous state space. Continuous methods are described further in Chapter 7.

Learning a model of a continuous domain requires predictions about a continuous next-state and reward from a continuous state. Unlike the discrete case, one cannot simply learn a tabular model for some discrete set of states. Some form of function approximation must be used, as many real-valued states may never be visited. Common approaches are to use regression or instance-based techniques to learn a continuous model.

A more difficult problem is to plan over a continuous state space. There are an infinite number of states for which the agent needs to know an optimal action. Again, this can be done with some form of function approximation on the policy, or the statespace could be discretized for planning purposes (even if used as-is for learning the model). In addition, many of the model-free approaches for continuous state spaces discussed in Chapter 7, such as policy gradient methods (Sutton et al, 1999) or Q-LEARNING with function approximation, could be used for planning a policy on the model.

Fitted value iteration (FVI) (Gordon, 1995) adapts value iteration to continuous state spaces. It iterates, updating the values of a finite set of states sampled from the infinite state space and then fitting a function approximator to their values. If the function approximator fits some contraction criteria, then fitted value iteration is proven to converge.

One of the earliest model-based RL algorithms for continuous state-spaces is the PARTI-GAME algorithm (Moore and Atkeson, 1995). It does not work in the typical RL framework, having deterministic dynamics and a goal region rather than a reward function. The algorithm discretizes the state space for learning and planning, adaptively increasing its resolution in interesting parts of the state space. When the planner is unable to find a policy to the goal, cells on the border of ones that succeed and fail are split further to increase the resolution of the discretization. This approach allows the agent to have a more accurate model of dynamics when needed and a more general model elsewhere.

Unlike the partitioning approach of the PARTI-GAME algorithm, Ormnoneit and Sen (2002) use an instance-based model of the domain in their kernel-based RL algorithm. The algorithm saves all the transitions it has experienced. When making a prediction for a queried state-action, the model makes a prediction based on an average of nearby transitions, weighted using the kernel function. This model is combined with approximate dynamic programming to create a full model-based method.

Deisenroth and Rasmussen (2011) use Gaussian Process (GP) regression to learn a model of the domain in their algorithm called Probabilistic Inference for Learning Control (PILCO). The GP regression model generalizes to unseen states and provides confidence bounds for its predictions. The agent plans assuming the next state

distribution matches the confidence bounds, encouraging the agent to explore when some states from the next state distribution are highly valued. The algorithm also uses GP regression to represent its policy and it computes the policy with policy iteration. It runs in batch mode, alternatively taking batches of actions in the world and then re-computing its model and policy. The algorithm learns to control a physical cart-pole device with few samples, but pauses for 10 minutes of computation after every 2.5 seconds of action.

Jong and Stone (2007) present an algorithm called FITTED-R-MAX that extends R-MAX to continuous domains using an instance-based model. When a state-action is queried, the algorithm uses the nearest instances to the queried state-action. They use the relative effects of the nearest instances to predict the relative change in state that will occur for the queried state-action. Their algorithm can provide a distribution over next states which is then used for planning with fitted value iteration. The agent is encouraged to explore parts of the state space that do not have enough instances with an R-MAX type exploration bonus.

Least-Squares Policy Iteration (LSPI) (Lagoudakis and Parr, 2003) is a popular method for planning with function approximation (described further in Chapter 3). It performs approximate policy iteration when using linear function approximation. It calculates the policy parameters that minimize the least-squares difference from the Bellman equation for a given set of experiences. These experiences could come from a generative model or from saved experiences of the agent. However, LSPI is usually used for batch learning with experiences gathered through random walks because of the expensive computation required. Li et al (2009) extend LSPI to perform online exploration by providing exploration bonuses similar to FITTED-R-MAX.

Nouri and Littman (2010) take a different approach with a focus on exploration in continuous domains. They develop an algorithm called Dimension Reduction in Exploration (DRE) that uses a method for dimensionality reduction in learning the transition function that automatically discovers the relevant state features for prediction. They predict each feature independently, and they use a ‘knownness’ criterion from their model to drive exploration. They combine this model with fitted value iteration to plan every few steps.

While RL typically focuses on discrete actions, there are many control problems that require a continuous control signal. Trying to find the best action in this case can be a difficult problem. Binary Action Search (Pazis and Lagoudakis, 2009) provides a possible solution to this problem by discretizing the action space and breaking down the continuous action selection problem into a series of binary action selections, each one deciding one bit of the value of the continuous action to be taken. Alternatively, one can represent the policy with a function approximator (for example, in an actor-critic method) and update the function approximator appropriately to output the best continuous action (Sutton et al, 1999; van Hasselt and Wiering, 2007). Weinstein et al (2010) develop a different approach called HOOT, using MCTS-type search to partition and search the continuous action space. The search progresses through the continuous action space, selecting smaller and smaller partitions of the continuous space until reaching a leaf in the search tree and selecting an action.

4.9 Empirical Comparisons

Having surveyed the model-based approaches along the dimensions of how they combine planning and model learning, how they explore, and how they deal with continuous spaces, we now present some brief representative experiments that illustrate their most important properties. Though model-based methods are most useful in large domains with limited action opportunities (see Section 4.10), we can illustrate their properties on simple toy domains⁵.

We compared R-MAX (described in Section 4.5) with Q-LEARNING, a typical model-free method, on the Taxi domain. R-MAX was run with the number of visits required for a state to be considered known, M , set to 5. Q-LEARNING was run with a learning rate of 0.3 and ϵ -greedy exploration with $\epsilon = 0.1$. Both methods were run with a discount factor of 0.99. The Taxi domain (Dietterich, 1998), shown in Figure 4.9a, is a 5×5 gridworld with four landmarks that are labeled with one of the following colors: *red*, *green*, *blue* or *yellow*. The agent's state consists of its location in the gridworld in x, y coordinates, the location of the passenger (at a landmark or in the *taxi*), and the passenger's destination (a landmark). The agent's goal is to navigate to the passenger's location, pick the passenger up, navigate to the passenger's destination and drop the passenger off. The agent has six actions that it can take. The first four (*north*, *south*, *west*, *east*) move the agent to the square in that respective direction with probability 0.8 and in a perpendicular direction with probability 0.1. If the resulting direction is blocked by a wall, the agent stays where it is. The fifth action is the *pickup* action, which picks up the passenger if she is at the taxi's location. The sixth action is the *putdown* action, which attempts to drop off the passenger. Each of the actions incurs a reward of -1 , except for unsuccessful *pickup* or *putdown* actions, which produce a reward of -10 . The episode is terminated by a successful *putdown* action, which provides a reward of $+20$. Each episode starts with the passenger's location and destination selected randomly from the four landmarks and with the agent at a random location in the gridworld.

Figure 4.9b shows a comparison of the average reward accrued by Q-LEARNING and R-MAX on the Taxi domain, averaged over 30 trials. R-MAX receives large negative rewards early as it explores all of its 'unknown' states. This exploration, however, leads it to find the optimal policy faster than Q-LEARNING.

Next, we show the performance of a few model-based methods and Q-LEARNING on the Cart-Pole Balancing task. Cart-Pole Balancing is a continuous task, shown in Figure 4.10a, where the agent must keep the pole balanced while keeping the cart on the track. The agent has two actions, which apply a force of 10 N to the cart in either direction. Uniform noise between -5 and 5 N is added to this force. The state is made up of four features: the pole's position, the pole's velocity, the cart's position, and the cart's velocity. The agent receives a reward of $+1$ each time step until the episode ends. The episode ends if the pole falls, the cart goes off the track, or 1,000 time steps have passed. The task was simulated at 50 Hz. For the

⁵ Source code to re-create these experiments is available at:
http://www.ros.org/wiki/reinforcement_learning

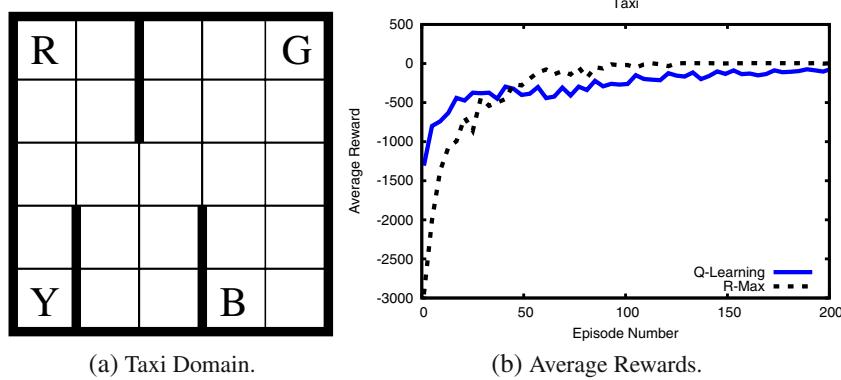


Fig. 4.9 4.9a shows the Taxi domain. 4.9b shows the average reward per episode for Q-LEARNING and R-MAX on the Taxi domain averaged over 30 trials.

discrete methods, we discretized each of the 4 dimensions into 10 values, for a total of 10,000 states.

For model-free methods, we compared Q-LEARNING on the discretized domain with Q-LEARNING using tile-coding for function approximation on the continuous representation of the task. Both methods were run with a learning rate of 0.3 and ϵ -greedy exploration with $\epsilon = 0.1$. Q-LEARNING with tile coding was run with 10 conjunctive tilings each with dimension split into 4 tiles. We also compared four model-based methods: two discrete methods and two continuous methods. The discrete methods were R-MAX (Brafman and Tennenholtz, 2001), which uses a tabular model, and TEXPLORE (Hester and Stone, 2010). TEXPLORE uses decision trees to model the relative effects of transitions and acts greedily with respect to a model that is the average of multiple possible models. Both of these methods were run on the discretized version of the domain. Here again, R-MAX was run with $M = 5$. TEXPLORE was run with $b = 0$, $w = 0.55$, and $f = 0.2$. We also evaluated FITTED-R-MAX (Jong and Stone, 2007), which is an extension of R-MAX for continuous domains, and CONTINUOUS TEXPLORE, an extension of TEXPLORE to continuous domains that uses regression trees to model the continuous state instead of discrete decision trees. CONTINUOUS TEXPLORE was run with the same parameters as TEXPLORE and FITTED-R-MAX was run with a model breadth of 0.05 and a resolution factor of 4.

The average rewards for the algorithms averaged over 30 trials are shown in Figure 4.10b. Both versions of R-MAX take a long time exploring and do not accrue much reward. Q-LEARNING with tile coding out-performs discrete Q-LEARNING because it is able to generalize values across states to learn faster. Meanwhile, the generalization and exploration of the TEXPLORE methods give them superior performance, as they accrue significantly more reward per episode than the other methods after episode 6. For each method, the continuous version of the algorithm out-performs the discrete version.

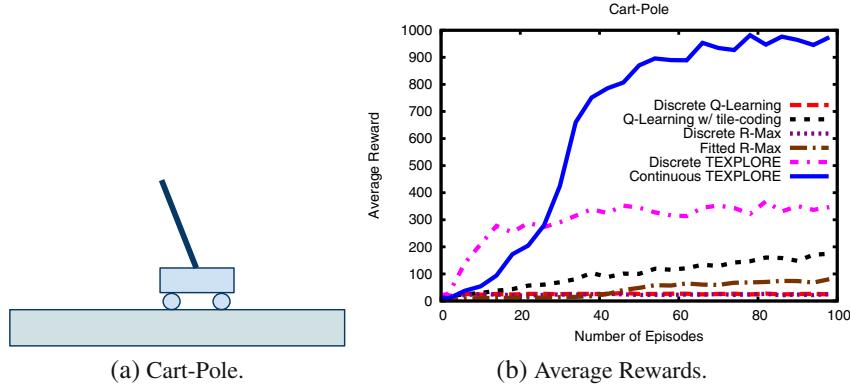


Fig. 4.10 4.10a shows the Cart-Pole Balancing task. 4.10b shows the average reward per episode for the algorithms on Cart-Pole Balancing averaged over 30 trials.

These experiments illustrate some of the trade-offs between model-based and model-free methods and among different model-based methods. Model-based methods such as R-MAX spend a long time exploring the domain in order to guarantee convergence to an optimal policy, allowing other methods such as Q-LEARNING to perform better than R-MAX during this exploration period. On the other hand, methods such as TEXPLORE that attempt to learn a model without exploring as thoroughly as R-MAX can learn to act reasonably well quickly, but do not provide the same guarantees of optimality since their model may not be completely accurate. In addition, when working in continuous domains, algorithms that work directly with the continuous variables without requiring discretization have a distinct advantage.

4.10 Scaling Up

While model-based RL provides great promise in terms of sample efficiency when compared to model-free RL, this advantage comes at the expense of more computational and space complexity for learning a model and planning. For this reason, evaluation of these methods has often been limited to small toy domains such as Mountain Car, Taxi, or Puddle World. In addition, even efficient algorithms may take too many actions to be practical on a real task. All of these issues must be resolved for model-based RL to be scaled up to larger and more realistic problems.

With factored models and generalization, model-based RL has achieved the sample efficiency to work on larger problems. However, in these cases, it still needs to plan over the entire state space when its model changes, requiring significant computation. One way that this problem has been addressed in the literature is to combine these methods with sample-based planners such as UCT (Kocsis and Szepesvári, 2006). In (Walsh et al., 2010), the authors prove that their method's bounds on

sample complexity are still valid when using a sample-based planner called Forward-Search Sparse Sampling (FSSS), which is a more conservative version of UCT. FSSS maintains statistical upper and lower bounds for the value of each node in the tree and only explores sub-trees that have a chance of being optimal.

In order to guarantee that they will find the optimal policy, any model-based algorithm must visit at least every state-action in the domain. Even this number of actions may be too large in some cases, whether it is because the domain is very big, or because the actions are very expensive or dangerous. These bounds can be improved in factored domains by assuming a DBN transition model. By assuming knowledge of the DBN structure ahead of time, the DBN- E^3 and FACTORED-R-MAX algorithms (Kearns and Koller, 1999; Guestrin et al, 2002) are able to learn a near-optimal policy in a number of actions polynomial in the number of parameters of the DBN-MDP, which can be exponentially smaller than the number of total states.

Another approach to this problem is to attempt to learn the structure of the DBN model using decision trees, as in the TEXPLORE algorithm (Hester and Stone, 2010). This approach gives up guarantees of optimality as the correct DBN structure may not be learned, but it can learn high-rewarding (if not optimal) policies in many fewer actions. The TEXPLORE algorithm models the MDP with random forests and explores where its models are uncertain, but does not explore every state-action in the domain.

Another approach to improving the sample efficiency of such algorithms is to incorporate some human knowledge into the agent. One approach for doing so is to provide trajectories of human generated experiences that the agent can use for building its model. For example, in (Ng et al, 2003), the authors learn a dynamics model of a remote control helicopter from data recorded from an expert user. Then they use a policy search RL method to learn a policy to fly the helicopter using their learned model.

One more issue with real-world decision making tasks is that they often involve partially-observable state, where the agent can not uniquely identify its state from its observations. The U-TREE algorithm (McCallum, 1996) is one model-based algorithm that addresses this issue. It learns a model of the domain using a tree that can incorporate previous states and actions into the splits in the tree, in addition to the current state and action. The historical states and actions can be used to accurately determine the agent's true current state. The algorithm then uses value iteration on this model to plan a policy.

Another way to scale up to larger and more complex domains is to use relational models (described further in Chapter 8). Here, the world is represented as a set of literals and the agent can learn models in the form of STRIPS-like planning operators (Fikes and Nilsson, 1971). Because these planning operators are relational, the agent can easily generalize the effects of its actions over different objects in the world, allowing it to scale up to worlds with many more objects. Pasula et al (2004) present a method for learning probabilistic relational models in the form of sets of rules. These rules describe how a particular action affects the state. They start with a set of rules based on the experiences the agent has seen so far, and perform a search over the rule set to optimize a score promoting simple and general rules.

Table 4.2 This table presents a summary of the algorithms presented in this chapter

Algorithm	Section	Model Type	Key Feature
DHP	4.2	Neural Network	Uses NN model within actor-critic framework
LWR RL	4.2	Locally Weighted Regression	One of the first methods to generalize model across states
DYNA	4.4	Tabular	Bellman updates on actual actions and saved experiences
Prioritized Sweeping	4.4	Tabular	Performs sweep of value updates backwards from changed state
DYNA-2	4.4	Tabular	Use UCT for simulated updates to transient memory
Real-Time Architecture	4.4	Any	Parallel architecture to allow real-time action
E^3	4.5	Tabular	Explicit decision to explore or exploit
R-MAX	4.5	Tabular	Reward bonus given to ‘unknown’ states
DBN- E^3	4.6	Learn probabilities for provided DBN model	Leans in number of actions polynomial in # of DBN parameters
FACTORED-R-MAX	4.6	Learn probabilities for provided DBN model	Leans in number of actions polynomial in # of DBN parameters
SLF-R-MAX	4.6	Learn models for all possible DBN structures	Explore when any model is uncertain or models disagree
MET-R-MAX	4.6	Learn DBN structure and probabilities efficiently	Explore when any model is uncertain or models disagree
LSE-R-MAX	4.6	Learn DBN efficiently without in-degree	Explore when any model is uncertain or models disagree
SPITI	4.6	Decision Trees	Model generalizes across states
RL-DT	4.6	Decision Trees with relative effects	Model generalizes across states, R-MAX-like exploration
Optimal Probe	4.7	Maintain distribution over models	Plan over augmented belief state space
BEETLE	4.7	Maintain distribution over parametrized models	Plan over augmented belief state space
Bayesian DP	4.7	Maintain distribution over models	Plan using model sampled from distribution
BOSS	4.7	Maintain distribution over models	Plan using merged model created from sampled models
MBBE	4.7	Maintain distribution over models	Use distribution over action-values to compute VPI
BEB	4.7	Maintain Dirichlet distribution over models	Provide reward bonus to follow Bayesian policy
MBIE	4.7	Maintain distribution over transition probabilities	Take max over transition probabilities as well as next state
TEXPLORE	4.7	Random Forest model for each feature	Plan approximate optimal policy on average model
Intrinsic Curiosity	4.7	Supervised Learning method	Provide intrinsic reward based on improvement in model
R-LAC	4.7	Supervised Learning method	Select actions based on improvement in model
PILCO	4.8	Gaussian Process Regression	Plan using uncertainty in next state predictions
PARTI-GAME	4.8	Split state-space based on transition dynamics	Splits state space non-uniformly
Kernel-based RL	4.8	Instance-based model with kernel distance	Use kernel to make predictions based on similar states
FITTED-R-MAX	4.8	Instance-based model	Predict based on applying the relative effect of similar transitions
DRE	4.8	Dimensionality Reduction techniques	Works in high-dimensional state spaces
U-TREE	4.10	Tree model that uses histories	Makes predictions in partially observable domains
Relational	4.10	Relational rule sets	Leans STRIPS-like relational operators

Object-oriented RL (Diuk et al, 2008) takes a similar approach to relational RL, defining the world in terms of objects.

4.11 Conclusion

Model-based methods learn a model of the MDP on-line while interacting with the environment, and then plan using their approximate model to calculate a policy. If the algorithm can learn an accurate model quickly enough, model-based methods can be more sample efficient than model-free methods. With an accurate learned model, an optimal policy can be planned without requiring any additional experiences in the world. In addition, these approaches can use their model to plan out multi-step exploration policies, enabling them to perform more directed exploration than model-free methods.

Table 4.2 shows a summary of the model-based algorithms described in this chapter. R-MAX (Brafman and Tennenholtz, 2001) is one of the most commonly used model-based methods because of its theoretical guarantees and ease of use and implementation. However, MET-R-MAX (Diuk et al, 2009) and LSE-R-MAX (Chakraborty and Stone, 2011) are the current state of the art in terms of the bounds on sample complexity for factored domains. There are other approaches that perform as well or better without such theoretical guarantees, such as Gaussian Process RL (Deisenroth and Rasmussen, 2011; Rasmussen and Kuss, 2004) or TEXPLORE (Hester and Stone, 2010).

A major direction of current research in model-based reinforcement learning is on scaling methods up to domains with larger and continuous state and action spaces. Accomplishing this goal will require making algorithms more sample efficient, relying on more computationally efficient planners, and developing better methods for exploration.

Acknowledgements. This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (IIS-0917122), ONR (N00014-09-1-0658), and the Federal Highway Administration (DTFH61-07-H-00030).

References

- Asmuth, J., Li, L., Littman, M., Nouri, A., Wingate, D.: A Bayesian sampling approach to exploration in reinforcement learning. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, UAI (2009)
- Atkeson, C., Moore, A., Schaal, S.: Locally weighted learning for control. *Artificial Intelligence Review* 11, 75–113 (1997)
- Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2), 235–256 (2002)
- Baranes, A., Oudeyer, P.Y.: R-IAC: Robust Intrinsically Motivated Exploration and Active Learning. *IEEE Transactions on Autonomous Mental Development* 1(3), 155–169 (2009)
- Boutilier, C., Dearden, R., Goldszmidt, M.: Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121, 49–107 (2000)
- Brafman, R., Tennenholz, M.: R-Max - a general polynomial time algorithm for near-optimal reinforcement learning. In: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI), pp. 953–958 (2001)
- Breiman, L.: Random forests. *Machine Learning* 45(1), 5–32 (2001)
- Chakraborty, D., Stone, P.: Structure learning in ergodic factored MDPs without knowledge of the transition function's in-degree. In: Proceedings of the Twenty-Eighth International Conference on Machine Learning, ICML (2011)
- Dearden, R., Friedman, N., Andre, D.: Model based Bayesian exploration. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI), pp. 150–159 (1999)
- Degriz, T., Sigaud, O., Wuillemin, P.H.: Learning the structure of factored Markov Decision Processes in reinforcement learning problems. In: Proceedings of the Twenty-Third International Conference on Machine Learning (ICML), pp. 257–264 (2006)
- Deisenroth, M., Rasmussen, C.: PILCO: A model-based and data-efficient approach to policy search. In: Proceedings of the Twenty-Eighth International Conference on Machine Learning, ICML (2011)
- Dietterich, T.: The MAXQ method for hierarchical reinforcement learning. In: Proceedings of the Fifteenth International Conference on Machine Learning (ICML), pp. 118–126 (1998)
- Diuk, C., Cohen, A., Littman, M.: An object-oriented representation for efficient reinforcement learning. In: Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML), pp. 240–247 (2008)

- Diuk, C., Li, L., Leffler, B.: The adaptive-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In: Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML), p. 32 (2009)
- Duff, M.: Design for an optimal probe. In: Proceedings of the Twentieth International Conference on Machine Learning (ICML), pp. 131–138 (2003)
- Even-dar, E., Mansour, Y.: Learning rates for q-learning. *Journal of Machine Learning Research*, 1–25 (2001)
- Fikes, R., Nilsson, N.: Strips: A new approach to the application of theorem proving to problem solving. Tech. Rep. 43r, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, SRI Project 8259 (1971)
- Gordon, G.: Stable function approximation in dynamic programming. In: Proceedings of the Twelfth International Conference on Machine Learning, ICML (1995)
- Guestrin, C., Patrascu, R., Schuurmans, D.: Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. In: Proceedings of the Nineteenth International Conference on Machine Learning (ICML), pp. 235–242 (2002)
- van Hasselt, H., Wiering, M.: Reinforcement learning in continuous action spaces. In: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL), pp. 272–279 (2007)
- Hester, T., Stone, P.: Generalized model learning for reinforcement learning in factored domains. In: Proceedings of the Eight International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS (2009)
- Hester, T., Stone, P.: Real time targeted exploration in large domains. In: Proceedings of the Ninth International Conference on Development and Learning, ICDL (2010)
- Hester, T., Quinlan, M., Stone, P.: Generalized model learning for reinforcement learning on a humanoid robot. In: Proceedings of the 2010 IEEE International Conference on Robotics and Automation, ICRA (2010)
- Hester, T., Quinlan, M., Stone, P.: A real-time model-based reinforcement learning architecture for robot control. ArXiv e-prints 11051749 (2011)
- Jong, N., Stone, P.: Model-based function approximation for reinforcement learning. In: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS (2007)
- Kakade, S.: On the sample complexity of reinforcement learning. PhD thesis, University College London (2003)
- Kearns, M., Koller, D.: Efficient reinforcement learning in factored MDPs. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI), pp. 740–747 (1999)
- Kearns, M., Singh, S.: Near-optimal reinforcement learning in polynomial time. In: Proceedings of the Fifteenth International Conference on Machine Learning (ICML), pp. 260–268 (1998)
- Kearns, M., Mansour, Y., Ng, A.: A sparse sampling algorithm for near-optimal planning in large Markov Decision Processes. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI), pp. 1324–1331 (1999)
- Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
- Kolter, J.Z., Ng, A.: Near-Bayesian exploration in polynomial time. In: Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML), pp. 513–520 (2009)

- Lagoudakis, M., Parr, R.: Least-squares policy iteration. *Journal of Machine Learning Research* 4, 1107–1149 (2003)
- Li, L., Littman, M., Walsh, T.: Knows what it knows: a framework for self-aware learning. In: *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML)*, pp. 568–575 (2008)
- Li, L., Littman, M., Mansley, C.: Online exploration in least-squares policy iteration. In: *Proceedings of the Eight International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 733–739 (2009)
- McCallum, A.: Learning to use selective attention and short-term memory in sequential tasks. In: *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior* (1996)
- Moore, A., Atkeson, C.: Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning* 13, 103–130 (1993)
- Moore, A., Atkeson, C.: The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning* 21, 199–233 (1995)
- Ng, A., Kim, H.J., Jordan, M., Sastry, S.: Autonomous helicopter flight via reinforcement learning. In: *Advances in Neural Information Processing Systems (NIPS)*, vol. 16 (2003)
- Nouri, A., Littman, M.: Dimension reduction and its application to model-based exploration in continuous spaces. *Mach. Learn.* 81(1), 85–98 (2010)
- Ormoneit, D., Sen, Š.: Kernel-based reinforcement learning. *Machine Learning* 49(2), 161–178 (2002)
- Pasula, H., Zettlemoyer, L., Kaelbling, L.P.: Learning probabilistic relational planning rules. In: *Proceedings of the 14th International Conference on Automated Planning and Scheduling, ICAPS* (2004)
- Pazis, J., Lagoudakis, M.: Binary action search for learning continuous-action control policies. In: *Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML)*, p. 100 (2009)
- Poupart, P., Vlassis, N., Hoey, J., Regan, K.: An analytic solution to discrete Bayesian reinforcement learning. In: *Proceedings of the Twenty-Third International Conference on Machine Learning (s)*, pp. 697–704 (2006)
- Prokhorov, D., Wunsch, D.: Adaptive critic designs. *IEEE Transactions on Neural Networks* 8, 997–1007 (1997)
- Rasmussen, C., Kuss, M.: Gaussian processes in reinforcement learning. In: *Advances in Neural Information Processing Systems (NIPS)*, vol. 16 (2004)
- Schaal, S., Atkeson, C.: Robot juggling: implementation of memory-based learning. *IEEE Control Systems Magazine* 14(1), 57–71 (1994)
- Schmidhuber, J.: Curious model-building control systems. In: *Proceedings of the International Joint Conference on Neural Networks*, pp. 1458–1463. IEEE (1991)
- Silver, D., Sutton, R., Müller, M.: Sample-based learning and search with permanent and transient memories. In: *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML)*, pp. 968–975 (2008)
- Strehl, A., Littman, M.: A theoretical analysis of model-based interval estimation. In: *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML)*, pp. 856–863 (2005)
- Strehl, A., Diuk, C., Littman, M.: Efficient structure learning in factored-state MDPs. In: *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pp. 645–650 (2007)
- Strens, M.: A Bayesian framework for reinforcement learning. In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pp. 943–950 (2000)

- Sutton, R.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings of the Seventh International Conference on Machine Learning (ICML), pp. 216–224 (1990)
- Sutton, R.: Dyna, an integrated architecture for learning, planning, and reacting. SIGART Bulletin 2(4), 160–163 (1991)
- Sutton, R., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems (NIPS), vol. 12, pp. 1057–1063 (1999)
- Venayagamoorthy, G., Harley, R., Wunsch, D.: Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator. IEEE Transactions on Neural Networks 13(3), 764–773 (2002)
- Walsh, T., Goschin, S., Littman, M.: Integrating sample-based planning and model-based reinforcement learning. In: Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (2010)
- Wang, T., Lizotte, D., Bowling, M., Schuurmans, D.: Bayesian sparse sampling for on-line reward optimization. In: Proceedings of the Twenty-Second International Conference on Machine Learning (ICML), pp. 956–963 (2005)
- Wang, Y., Gelly, S.: Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In: IEEE Symposium on Computational Intelligence and Games (2007)
- Weinstein, A., Mansley, C., Littman, M.: Sample-based planning for continuous action Markov Decision Processes. In: ICML 2010 Workshop on Reinforcement Learning and Search in Very Large Spaces (2010)
- Wiering, M., Schmidhuber, J.: Efficient model-based exploration. In: From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior, pp. 223–228. MIT Press, Cambridge (1998)

Chapter 5

Transfer in Reinforcement Learning: A Framework and a Survey

Alessandro Lazaric

Abstract. Transfer in reinforcement learning is a novel research area that focuses on the development of methods to transfer knowledge from a set of source tasks to a target task. Whenever the tasks are *similar*, the transferred knowledge can be used by a learning algorithm to solve the target task and significantly improve its performance (e.g., by reducing the number of samples needed to achieve a nearly optimal performance). In this chapter we provide a formalization of the general transfer problem, we identify the main settings which have been investigated so far, and we review the most important approaches to transfer in reinforcement learning.

5.1 Introduction

The idea of transferring knowledge across different but related tasks to improve the performance of machine learning (ML) algorithms stems from psychology and cognitive science research. A number of psychological studies (see e.g., Thorndike and Woodworth, 1901; Perkins et al, 1992) show that humans are able to learn a task better and faster by transferring the knowledge retained from solving similar tasks. Transfer in machine learning has the objective to design *transfer* methods that analyze the knowledge collected from a set of source tasks (e.g., samples, solutions) and transfer it so as to *bias* the learning process on a target task towards a set of *good* hypotheses. If the transfer method successfully identifies the similarities between source and target tasks, then the transferred knowledge is likely to improve the learning performance on the target task. The idea of retaining and reusing knowledge to improve the learning algorithms dates back to early stages of ML. In fact, it is widely recognized that a good representation is the most critical aspect of any learning algorithm, and the development of techniques that automatically change the representation according to the task at hand is one of the main objectives of

Alessandro Lazaric
INRIA Lille-Nord Europe, 40 Avenue Halley, 59650 Villeneuve d'Ascq, France
e-mail: alessandro.lazaric@inria.fr

large part of the research in ML. Most of the research in transfer learning (Fawcett et al, 1994) identified the single-problem perspective usually adopted in ML as a limit for the definition of effective methods for the inductive construction of good representations. On the other hand, taking inspiration from studies in psychology and neuroscience (Gentner et al, 2003; Gick and Holyoak, 1983), the transfer point of view, where learning tasks are assumed to be related and knowledge is retained and transferred, is considered as the most suitable perspective to design effective techniques of inductive bias (Utgoff, 1986).

Transfer in reinforcement learning. Transfer algorithms have been successful in improving the performance of learning algorithms in a number of supervised learning problems, such as recommender systems, medical decision making, text classification, and general game playing. In recent years, the research on transfer also focused on the reinforcement learning (RL) paradigm and how RL algorithms could benefit from knowledge transfer. In principle, traditional reinforcement learning already provides mechanisms to learn solutions for any task without the need of human supervision. Nonetheless, the number of samples needed to learn a nearly-optimal solution is often prohibitive in real-world problems unless prior knowledge from a domain expert is available. Furthermore, every time the task at hand changes the learning process must be restarted from scratch even when similar problems have been already solved. Transfer algorithms automatically build prior knowledge from the knowledge collected in solving a set of similar source tasks (i.e., *training* tasks) and use it to bias the learning process on any new task (i.e., *testing* task). The result is a dramatic reduction in the number of samples and a significant improvement in the accuracy of the learned solution.

Aim of the chapter. Unlike supervised learning, reinforcement learning problems are characterized by a large number of elements such as the dynamics and the reward function, and many different transfer settings can be defined depending on the differences and similarities between the tasks. Although relatively recent, research on transfer in reinforcement learning already counts a large number of works covering many different transfer problems. Nonetheless, it is often difficult to have a clear picture of the current state-of-the-art in transfer in RL because of the very different approaches and perspectives adopted in dealing with this complex and challenging problem. The aim of this chapter is to formalize what the main transfer settings are and to classify the algorithmic approaches according to the kind of knowledge they transfer from source to target tasks. Taylor and Stone (2009) also provide a thorough survey of transfer in reinforcement learning. While their survey provides a very in-depth analysis of each transfer algorithm, the objective of this chapter is not to review all the algorithms available in the literature but rather to identify the characteristics shared by the different approaches of transfer in RL and classify them into large families.

Structure of the chapter. The rest of the chapter is organized as follows. In Section 5.2 we formalize the transfer problem and we identify three main dimensions to categorize the transfer algorithms according to the setting, the transferred knowledge, and the objective. Then we review the main approaches of transfer in RL in

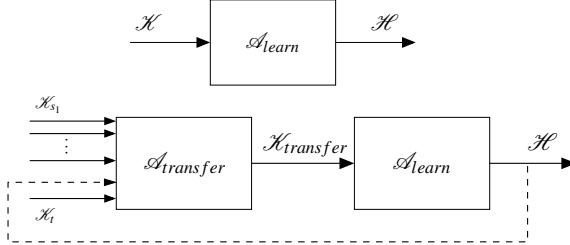


Fig. 5.1 (top) In the standard learning process, the learning algorithm gets as input some form of knowledge about the task (i.e., samples, structure of the solutions, parameters) and returns a solution. **(bottom)** In the transfer setting, a transfer phase first takes as input the knowledge retained from a set of source tasks and returns a new knowledge which is used as input for the learning algorithm. The dashed line represents the possibility to define a continual process where the experience obtained from solving a task is then reused in solving new tasks.

three different settings. In Section 5.3 we focus on the source-to-target setting where transfer occurs from one single source task to one single target task. A more general setting with a set of source tasks and one target task is studied in Section 5.4. Finally, in Section 5.5 we discuss the general source-to-target setting when the state-action spaces of source and target tasks are different. In Section 5.6 we conclude and we discuss open questions.

5.2 A Framework and a Taxonomy for Transfer in Reinforcement Learning

Transfer learning is a general problem and it is difficult to provide a formal definition able to take into account all the possible perspectives and approaches to the problem. Furthermore, although many different algorithms have been already proposed, a clear categorization of the main approaches to transfer in RL is still missing. In this section we first introduce a formalization of the general transfer and we then propose a taxonomy to classify transfer approaches along three main dimensions.

5.2.1 Transfer Framework

In this section, we adapt the formalisms introduced by Baxter (2000) and Silver (2000) for supervised learning to the RL paradigm and we introduce general definitions and symbols used throughout the rest of the chapter.

As discussed in the introduction, transfer learning leverages on the knowledge collected from a number of different tasks to improve the learning performance in new tasks. We define a *task M* as an MDP (Sutton and Barto, 1998) characterized

Table 5.1 List of the main symbols used in the chapter

Symbol	Meaning
M	MDP
S_M	State space
A_M	Action space
T_M	Transition model (dynamics)
R_M	Reward function
\mathcal{M}	Task space (set of tasks M)
Ω	Probability distribution over \mathcal{M}
\mathcal{E}	Environment (task space and distribution)
\mathcal{H}	Hypothesis space (e.g., value functions, policies)
$h \in \mathcal{H}$	Hypothesis (e.g., one value function, one policy)
\mathcal{K}	Knowledge space (e.g., samples and basis functions)
$K \in \mathcal{K}$	Knowledge (e.g., specific realization of samples)
\mathcal{K}_s	Knowledge space from a source task
\mathcal{K}_t	Knowledge space from a target task
$\mathcal{K}_{\text{transfer}}$	Knowledge space return by the transfer algorithm and used in learning
\mathcal{F}	Space of functions defined on a specific state-action space
ϕ	State-action basis function
$\mathcal{A}_{\text{learn}}$	Learning algorithm
$\mathcal{A}_{\text{transfer}}$	Transfer algorithm
O	Set of options
$o \in O$	An option

by the tuple $\langle S_M, A_M, T_M, R_M \rangle$ where S_M is the state space, A_M is the action space, T_M is the transition function, and R_M is the reward function. While the state-action space $S_M \times A_M$ defines the *domain* of the task, the transition T_M and reward function R_M define the *objective* of the task. The *space of tasks* involved in the transfer learning problem is denoted by $\mathcal{M} = \{M\}$. Let Ω be a probability distribution over the space of tasks \mathcal{M} , then we denote by $\mathcal{E} = \langle \mathcal{M}, \Omega \rangle$ the *environment*, which defines the setting of the transfer problem. The tasks presented to the learner are drawn from the task distribution (i.e., $M \sim \Omega$). This general definition resembles the traditional supervised learning setting where training samples are drawn from a given distribution. As a result, similar to classification and regression, transfer learning is based on the idea that since tasks are drawn from the same distribution, an algorithm able to achieve a good performance on average on a finite number of source tasks (or training tasks), then it will also generalize well across the target tasks in \mathcal{M} coming from the same distribution Ω (or testing tasks).

A standard learning algorithm takes as input some form of knowledge of the task at hand and returns a solution in a set of possible results. We use \mathcal{K} to denote the space of the knowledge used as input for the learning algorithm and \mathcal{H} for the space of hypotheses that can be returned. In particular, \mathcal{K} refers to all the elements used by the algorithm to compute the solution of a task, notably the *instances* (e.g.,

samples), the *representation* of the problem (e.g., set of options, set of features), and *parameters* (e.g., learning rate) used by the algorithm. Notice that \mathcal{H} includes *prior* knowledge provided by an expert, *transfer* knowledge obtained from a transfer algorithm, and *direct* knowledge collected from the task. A general learning algorithm is defined as the mapping

$$\mathcal{A}_{\text{learn}} : \mathcal{H} \rightarrow \mathcal{H}. \quad (5.1)$$

Example 1. Let us consider fitted Q -iteration (Ernst et al, 2005) with a linear function approximator. Fitted Q -iteration first collects N samples (the *instances*) and through an iterative process returns an action-value function which approximates the optimal action-value function of the task. In this case, the hypothesis space \mathcal{H} is the linear space spanned by a set of d features $\{\varphi_i : S \times A \rightarrow \mathbb{R}\}_{i=1}^d$ designed by a domain expert, that is $\mathcal{H} = \{h(\cdot, \cdot) = \sum_{i=1}^d \alpha_i \varphi_i(\cdot, \cdot)\}$. Beside this prior knowledge, the algorithm also receives as input a set of N samples $\langle s, a, s', r \rangle$. As a result, the knowledge used by fitted Q -iteration can be formalized by the space $\mathcal{H} = ((S \times A \times S \times R)^N, \mathcal{F}^d)$, where any specific instance $K \in \mathcal{H}$ is $K = (\{\langle s_n, a_n, r_n, s'_n \rangle\}_{n=1}^N, \{\varphi_i\}_{i=1}^d)$, with $\varphi_i \in \mathcal{F}$. Given as input $K \in \mathcal{H}$ the algorithm returns an action-value function $h \in \mathcal{H}$ (i.e., $\mathcal{A}_{FQI}(K) = h$). \square

Given the previous definitions, we can now define the general shape of transfer learning algorithms. In general, in single-task learning only the instances are directly collected from the task at hand, while the representation of the problem and the parameters are given as a prior by an expert. In transfer learning, the objective is to reduce the need for instances from the target task and prior knowledge from a domain expert by tuning and adapting the structure of the learning algorithm (i.e., the knowledge used as input) on the basis of the previous tasks observed so far. Let $\mathcal{E} = \langle \mathcal{M}, \Omega \rangle$ be the environment at hand and L be the number of tasks drawn from \mathcal{M} according to the distribution Ω used as source tasks, a transfer learning algorithm is usually the result of a *transfer of knowledge* and a *learning* phase. Let \mathcal{H}_s^L be the knowledge collected from the L source tasks and \mathcal{H}_t the knowledge available (if any) from the target task. The transfer phase is defined as

$$\mathcal{A}_{\text{transfer}} : \mathcal{H}_s^L \times \mathcal{H}_t \rightarrow \mathcal{H}_{\text{transfer}}, \quad (5.2)$$

where $\mathcal{H}_{\text{transfer}}$ is the final knowledge transferred to the learning phase. In particular, the learning algorithm is now defined as

$$\mathcal{A}_{\text{learn}} : \mathcal{H}_{\text{transfer}} \times \mathcal{H}_t \rightarrow \mathcal{H}. \quad (5.3)$$

Example 2. Let us consider the transfer algorithm introduced by Lazaric (2008) in which a set of features is learned from a set of L source tasks. In this case $\mathcal{A}_{\text{transfer}}$ takes as input N_s samples for each of the L tasks and returns d features $\{\varphi_i\}_{i=1}^d$ from \mathcal{F} . The fitted Q -iteration algorithm is then used to learn the solution of a target task and $\mathcal{A}_{\text{learn}}$ takes as input N_t target samples and the features extracted during the transfer phase and returns a function in the space \mathcal{H} spanned by the

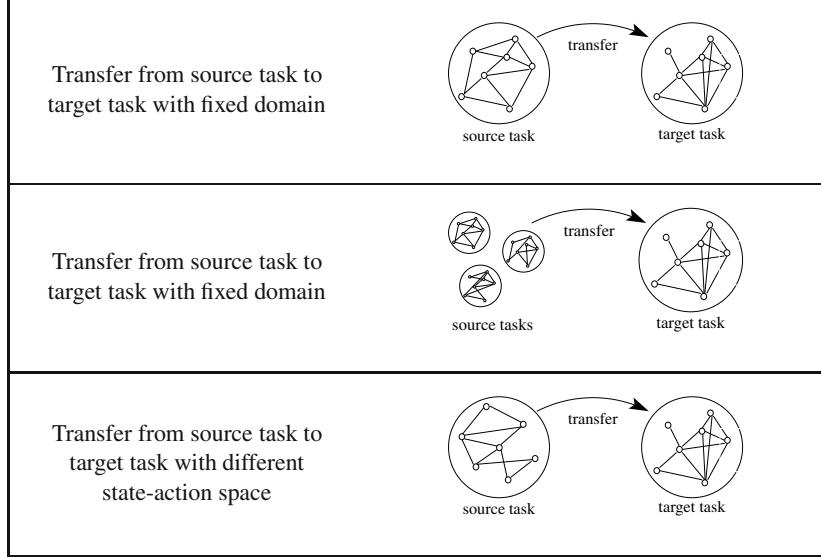


Fig. 5.2 The three main transfer settings defined according to the number of source tasks and their difference w.r.t. the target task

features $\{\varphi_i\}_{i=1}^d$. Thus, we have $\mathcal{K}_s = (S \times A \times S \times R)^{N_s}$, $\mathcal{K}_t = (S \times A \times S \times R)^{N_t}$, and $\mathcal{K}_{\text{transfer}} = \mathcal{F}^d$. \square

Although in the definition in Equation (5.2) \mathcal{K}_t is present in both the transfer and learning phase, in most of the transfer settings, no knowledge about the target is available in the transfer phase. This formalization also shows that transfer algorithms must be compatible with the specific learning algorithm employed in the second phase, since $\mathcal{K}_{\text{transfer}}$ is used as an additional source of knowledge for $\mathcal{A}_{\text{learn}}$. The performance of the transfer algorithm is usually compared to a learning algorithm in Equation (5.1) which takes as input only \mathcal{K}_t . As discussed in the next section, the specific setting \mathcal{E} , the knowledge spaces \mathcal{K} , and the way the performance is measured define the main categories of transfer problems and approaches.

5.2.2 Taxonomy

In this section we propose a taxonomy of the major approaches to transfer in reinforcement learning. We define three main dimensions: the *setting*, the *transferred knowledge*, and the *objective*.

5.2.2.1 The Settings

In the general formulation of the transfer problem we define an environment \mathcal{E} as the space of tasks \mathcal{M} and the probability distribution Ω on it. Unlike other learning paradigms (see Pan and Yang (2010) for a review of the possible settings in supervised learning), an RL problem is defined by different elements such as the dynamics and the reward, and the tasks in \mathcal{M} may differ in a number of possible ways depending on the similarities and differences in each of these elements. For instance, in the transfer problem considered by Mehta et al (2008) all the tasks share the same state-action space and dynamics but the reward functions are obtained as linear combinations of basis reward functions and a weight vector. In this case, the space of tasks \mathcal{M} is the set of MDPs which can be generated by varying the weights of the reward functions. Furthermore, although in the general definition tasks are drawn from a distribution Ω , there are many transfer settings in which the tasks are fixed in advance and no generalization over other tasks is considered. For instance, most of the inter-task mapping approaches (see e.g., Taylor et al, 2007a) focus on the setting in which only one source task and one target task are available. Although an implicit assumption of similarity is usually made, the tasks are simply given as input to the algorithm and no explicit distribution is defined.

In the following we will distinguish among three different categories of transfer settings (see Figure 5.2).

- (I) *Transfer from source task to target task with fixed domain.* As defined in Section 5.2.1 the domain of a task is determined by its state-action space $S_M \times A_M$, while the specific structure and goal of the task are defined by the dynamics T_M and reward R_M . Most of the early literature in transfer in RL focused on the setting in which the domain is fixed and only two tasks are involved: a *source* task and a *target* task. This setting is usually referred to *inductive transfer learning* in the supervised learning literature (Pan and Yang, 2010). The transfer algorithm might or might not have access to the target task at transfer time. If no target knowledge is available, some of the transfer algorithms perform a shallow transfer of the knowledge collected in the source task (e.g., the policy) and directly use it in the target task. Other algorithms try to abstract from the source task some general characteristics (e.g., subgoals) that are likely to be relevant in solving target tasks sharing the same characteristics. On the other hand, when some target knowledge is available at transfer time, then it is used to *adapt* the source knowledge to the target task. For instance, in (Taylor et al, 2008b) target samples are used to identify the best mapping between source and target state-action variables and thus to transform the source policy into a target policy used to initialize the learning process in the target task.
- (II) *Transfer across tasks with fixed domain.* In this setting, the general definition of environment \mathcal{E} with a distribution over the task space is considered. In this case, tasks share the same domain and the transfer algorithm takes as input the knowledge collected from a set of source tasks and use it to improve the performance in the target task. In this setting, the objective is usually to generalize over the tasks in \mathcal{M} according to the distribution Ω . Similar to supervised

learning, we expect that, as the number of source tasks increases, the transfer algorithm is able to improve the average performance on the target tasks drawn from Ω when compared to a single-task learning algorithm which does not use any transferred knowledge.

- (III) *Transfer across tasks with different domains.* Finally, in this setting tasks have a different domain, that is they might have different state-action variables, both in terms of number and range. Most of the transfer approaches in this case consider the source-target scenario and focus on how to define a mapping between the source state-action variables and the target variables so as to obtain an effective transfer of knowledge.

5.2.2.2 The Knowledge

The definition of transferred knowledge and the specific transfer process are the main aspects characterizing a transfer learning algorithm. In the definition of Section 5.2.1 the space \mathcal{K} contains the *instances* collected from the environment (e.g., sample trajectories), the *representation* of the solution and the *parameters* of the algorithm itself. Once the space of knowledge considered by the algorithm is defined, it is important to design how this knowledge is actually used to transfer information from the source tasks to the target task. Silver (2000) and Pan and Yang (2010) propose a general classification of the knowledge retained and transferred across tasks in supervised learning. Taylor and Stone (2009) introduces a very detailed classification for transfer in RL. Here we prefer to have a broader classification identifying macro-categories of approaches along the lines of Lazaric (2008). We classify the possible knowledge transfer approaches into three categories: *instance transfer*, *representation transfer*, *parameter transfer*.

- (I) *Instance transfer.* Unlike dynamic programming algorithms, where the dynamics and reward functions are known in advance, all the RL algorithms rely on a set of samples collected from a direct interaction with the MDP to build a solution for the task at hand. This set of samples can be used to estimate the model of the MDP in *model-based* approaches or to directly build an approximation of the value function or policy in *model-free* approaches. The most simple version of transfer algorithm collects samples coming from different source tasks and reuses them in learning the target task. For instance, the transfer of trajectory samples can be used to simplify the estimation of the model of new tasks (Sunmola and Wyatt, 2006) or the estimation of the action value function as in (Lazaric et al, 2008).
- (II) *Representation transfer.* Each RL algorithm uses a specific representation of the task and of the solution, such as state-aggregation, neural networks, or a set of basis functions for the approximation of the optimal value function. After learning on different tasks, transfer algorithms often perform an abstraction process which changes the representation of the task and of the solutions. In this category, many possible approaches are possible varying from reward

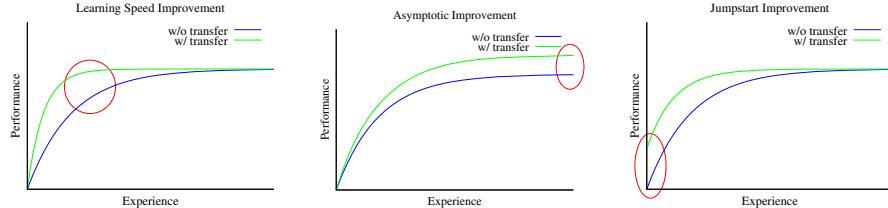


Fig. 5.3 The three main objectives of transfer learning (Langley, 2006). The red circles highlight the improvement in the performance in the learning process expected by using transfer solutions w.r.t. single-task approaches.

shaping (Konidaris and Barto, 2006) and MDP augmentation through options (Singh et al, 2004) to basis function extraction (Mahadevan and Maggioni, 2007).

(III) *Parameter transfer*: Most of the RL algorithms are characterized by a number of parameters which define the initialization and the behavior of the algorithm itself. For instance, in Q-learning (Watkins and Dayan, 1992) the Q-table is initialized with arbitrary values (e.g., the highest possible value for the action values $R_{max}/(1 - \gamma)$) and it is updated using a gradient-descent rule with a learning rate α . The initial values and the learning rate define the set of input parameters used by the algorithm. Some transfer approaches change and adapt the algorithm parameters according to the source tasks. For instance, if the action values in some state-action pairs are very similar across all the source tasks, the Q-table for the target task could be initialized to more convenient values thus speeding-up the learning process. In particular, the transfer of initial solutions (i.e., policies or value functions) is commonly adopted to initialize the learning algorithm in the transfer setting with only one source task.

5.2.2.3 The Objectives

While in supervised learning the performance of a classifier or a regressor are usually measured in terms of prediction error, in RL many possible measures can be employed to evaluate how good is the solution returned by the learning algorithm. As a result, transfer algorithms can be evaluated according to a number of different performance measures. Depending on how the learning performance is measured, different transfer metrics may be used. In Taylor and Stone (2009) a number of metrics is proposed to measure the improvement of transfer over single-task approaches. Here we discuss three main transfer objectives adapted from the objectives suggested for the general problem of transfer suggested by Langley (2006) (see Figure 5.3):

- (I) *Learning speed improvement*. This objective is about the reduction in the amount of the experience needed to learn the solution of the task at hand. As new tasks are sampled according to Ω , the knowledge retained from a set of

previously solved tasks can be used to bias the learning algorithm towards a limited set of solutions, so as to reduce its learning time. The complexity of a learning algorithm is usually measured by the number of samples needed to achieve a desired performance. In RL, this objective is pursued following two different approaches. The first approach is to make the algorithm more effective in using the experience collected from the exploration of the environment. For instance, Kalmar and Szepesvari (1999) and Hauskrecht (1998) show that the use of options can improve the effectiveness of value iteration backups by updating value function estimates with the total reward collected by an option, and thus reducing the number of iterations to converge to a nearly optimal solution. The second aspect is about the strategy used to collect the samples. In online RL algorithms samples are collected from direct interaction with the environment through an exploration strategy. The experience collected by solving a set of tasks can lead to the definition of better exploration strategies for new related tasks. For instance, if all the tasks have goals in a limited region of the state space, an exploration strategy that frequently visits that region will lead to more informative samples.

In practice, at least three different methods can be used to measure the improvement in the learning speed: *time to threshold*, *area ratio*, and *finite-sample analysis*. In all the problems where a target performance is considered (e.g., a small enough number of steps-to-go in a navigation problem), it is possible to set a threshold and measure how much experience (e.g., samples, episodes, iterations) is needed by the single-task and transfer algorithms to achieve that threshold. If the transfer algorithm successfully takes advantage of the knowledge collected from the previous tasks, we expect it to need much less experience to reach the target performance. The main drawback of this metric is that the threshold might be arbitrary and that it does not take into account the whole learning behavior of the algorithms. In fact, it could be the case that an algorithm is faster in reaching a given threshold but it has a very poor initial performance or does not achieve the asymptotic optimal performance. The area ratio metric introduced by Taylor and Stone (2009) copes with this problem by considering the whole area under the learning curves with and without transfer. Formally, the area ratio is defined as

$$r = \frac{\text{area with transfer} - \text{area without transfer}}{\text{area without transfer}}. \quad (5.4)$$

Although this metric successfully takes into consideration the behavior of the algorithms until a given number of samples, it is scale dependent. For instance, when the reward-per-episode is used as a measure of performance, the scale of the rewards impacts on the area ratio and changes in the rewards might lead to different conclusions in the comparison of different algorithms. While the two previous measures allow to empirically compare the learning performance with and without transfer, it is also interesting to have a more rigorous comparison by deriving sample-based bounds for the algorithms at hand. In such case, it is possible to compute an upper bound on the error of the solution returned by the

algorithm depending on the parameters of the task and the number of samples is available. For instance, if the algorithm returns a function $h \in \mathcal{H}$ and Q^* is the optimal action value function, a finite sample bound is usually defined as

$$\|h - Q^*\|_\rho \leq \varepsilon_1(\mathcal{H}, Q^*) + \varepsilon_2(N), \quad (5.5)$$

where ρ is a distribution over the state space S , $\varepsilon_1(\mathcal{H}, Q^*)$ is the *approximation* error and it accounts for the asymptotic error of the best possible solution in \mathcal{H} , and $\varepsilon_2(N)$ is the *estimation* error and it decreases with the number of samples. Transfer algorithms should be able to reduce the estimation error such that with the same number of samples as in the single-task algorithm, they could achieve a better performance. Although recent works in RL provide finite-sample analysis for a number of popular algorithms such as fitted value iteration (Munos and Szepesvári, 2008), LSTD (Farahmand et al, 2008; Lazaric et al, 2010), and Bellman residual minimization (Antos et al, 2008; Maillard et al, 2010), at the best of our knowledge, at the moment there is no finite-sample analysis for any transfer algorithm in RL.

As it will be reviewed in next sections, this objective is usually pursued by **instance-transfer** by adding source samples to the set of samples used to learn the target task and by **parameter-transfer approaches by initializing the learning process to a convenient solution**. Representation-transfer algorithms achieve a learning speed improvement by augmenting the current representation of the task (e.g., adding options to the action set) and of the solutions (i.e., adding features).

- (II) *Asymptotic improvement.* In most of the problems of practical interest, a perfect approximation of the optimal value function or policy is not possible (e.g., problems with continuous state-action spaces) and the use of function approximation techniques is mandatory. The more accurate the approximation, the better the generalization (and the performance) at convergence. The accuracy of the approximation is strictly dependent on the structure of the space of hypotheses \mathcal{H} used to represent the solution (e.g., value functions). This objective is usually targeted by representation-transfer algorithms which adapt the structure of \mathcal{H} (e.g., by changing the features in a linear approximation space) so as to accurately approximate the solutions of the tasks in \mathcal{M} . An empirical measure of the quality of \mathcal{H} is to compare the asymptotic performance (i.e., when a large number of samples is available) of transfer and single-task learning. Also in this case it would be interesting to analyze the effectiveness of the transfer algorithms by providing a finite-sample analysis of their performance. In particular, the asymptotic improvement corresponds to a better approximation error term in the bound of Equation (5.5). Similar to the learning speed improvement, at the moment no transfer algorithm is guaranteed to improve the average approximation error over the tasks in \mathcal{M} .
- (III) *Jumpstart improvement.* The learning process usually starts from either a random or an arbitrary hypothesis h in the hypothesis space \mathcal{H} . According to the definition of environment, all the tasks are drawn from the same distribution Ω .

Table 5.2 The three dimensions of transfer learning in RL. Each transfer solution is specifically designed for a *setting*, it transfers some form of *knowledge*, and it pursues an *objective*. The survey classifies the existing algorithms according to the first dimension, it then reviews the approaches depending on the transferred knowledge and discusses which objectives they achieve.



Setting	Knowledge	Objective
Transfer from source to target with fixed domain	Instance	Learning speed
Transfer across tasks with fixed domain	Representation	Asymptotic performance
Transfer across tasks with different domains	Parameter	Jumpstart

As a result, after observing a number of source tasks, the transfer algorithm may build an effective prior on the solution of the tasks in \mathcal{M} and initialize the learning algorithm to a suitable initial hypothesis with a better performance w.r.t. to a random initialization. It is worth noting that this objective does not necessarily correspond to an improvement in the learning speed. Let us consider a source task whose optimal policy is significantly different from the optimal policy of the target task but that, at the same time, it achieves only a slightly suboptimal performance (e.g., two goal states with different final positive rewards in different regions of the state space). In this case, the improvement of the initial performance can be obtained by initializing the learning algorithm to the optimal policy of the source task, but this may lead to worsen the learning speed. In fact, the initial policy does not provide samples of the actual optimal policy of the task at hand, thus slowing down the learning algorithm. On the other hand, it could be possible that the policy transferred from the source task is an effective exploration strategy for learning the optimal policy of the target task, but that it also achieves very poor performance. This objective is usually pursued by parameter-transfer algorithms in which the learning algorithm is initialized with a suitable solution whose performance is better compared to a random (or arbitrary) initialization.

5.2.2.4 The Survey

Given the framework introduced in the previous sections, the survey is organized along the dimensions in Table 5.2. In the following sections we first classify the main transfer approaches in RL according to the specific setting they consider. In each setting, we further divide the algorithms depending on the type of knowledge they transfer from source to target, and, finally, we discuss which objectives are achieved. As it can be noticed, the literature on transfer in RL is not equally distributed on the three settings. Most of the early literature on transfer in RL focused on the source-to-target setting, while the most popular scenario of recent research is the general problem of transfer from a set of source tasks. Finally, research on the problem of mapping different state and action spaces mostly relied on hand-coded transformations and much room for further investigation is available.

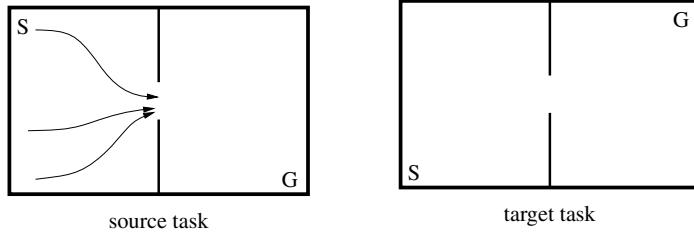


Fig. 5.4 Example of the setting of transfer from source to target with a fixed state-action space

5.3 Methods for Transfer from Source to Target with a Fixed State-Action Space

In this section we consider the most simple setting in which transfer occurs from one source task to a target task. We first formulate the general setting in the next section and we then review the main approaches to this problem by categorizing them according to the type of transferred knowledge. Most of the approaches reviewed in the following change the representation of the problem or directly transfer the source solution to the target task. Furthermore, unlike the other two settings considered in Section 5.4 and 5.5, not all the possible knowledge transfer models are considered and at the best of our knowledge no instance-transfer method has been proposed for this specific setting.

5.3.1 Problem Formulation

In this transfer setting we define two MDPs, a *source* task $M_s = \langle S, A, T_s, R_s \rangle$ and a *target* task $M_t = \langle S, A, T_t, R_t \rangle$, sharing the same state-action space $S \times A$. The environment \mathcal{E} is defined by the task space $\mathcal{M} = \{M_s, M_t\}$ and a task distribution Ω which simply returns M_s as the first task and M_t as second.

Example 3. Let us consider the transfer problem depicted in Figure 5.4. The source task is a navigation problem where the agent should move from the region marked with S to the goal region G . The target task shares exactly the same state-action space and the same dynamics as the source task but the initial state and the goal (and the reward function) are different. The transfer algorithm first collect some form of knowledge from the interaction with the source task and then generates a transferrable knowledge that can be used as input to the learning algorithm on the target task. In this example, the transfer algorithm can exploit the similarity in the dynamics and identify regularities that could be useful in learning the target task.

As reviewed in the next section, one effective way to perform transfer in this case is to discover policies (i.e., options) useful to navigate in an environment with such a dynamics. For instance, the policy sketched in Figure 5.4 allows the agent to move from any point in the left room to the door between the two rooms. Such a policy is useful to solve any navigation task requiring the agent to move from a starting region in the left room to a goal region in the right room. Another popular approach is to discover features which are well-suited to approximate the optimal value functions in the environment. In fact, the dynamics displays symmetries and discontinuities which are likely to be preserved in the value functions. For instance, both the source and target value functions are discontinuous close to the walls separating the two rooms. As a result, once the source task is solved, the transfer algorithm should analyze the dynamics and the value function and return a set of features which capture this discontinuity and preserve the symmetries of the problem. \square

5.3.2 Representation Transfer

In some transfer problems no knowledge about the target task is available before transfer actually takes place and \mathcal{H}_t in Equation (5.2) is always empty. In this case, it is important to abstract from the source task general characteristics that are likely to apply to the target task as well. The transfer algorithm first collects some knowledge from the source task and it then changes the representation either of the solution space \mathcal{H} or of the MDP so as to speed-up the learning in the target task.



Option discovery. One of the most popular approaches to the source-target transfer problem is to change the representation of the MDP by adding *options* (Sutton et al, 1999) to the set of available actions (see Chapter 9 for a review of hierarchical RL methods). In discrete MDPs, options do not affect the possibility to achieve the optimal solution (since all the primitive actions are available, any possible policy can still be represented), but they are likely to improve the learning speed if they reach regions of the state space which are useful to learn the target task. All the option-transfer methods consider discrete MDPs, a tabular representation of the action-value function, and source and target tasks which differ only in the reward function (i.e., $T_s = T_t$). The idea is to exploit the structure of the dynamics shared by the two tasks and to ignore the details about the specific source reward function. Most of these methods share a common structure. A set of samples $\langle s_i, a_i, r_i, s'_i \rangle$ is first collected from the source task and an estimated MDP \hat{M}_s is computed. On the basis of the characteristics of the estimated dynamics a set of relevant subgoals is identified and a set of d options is learned to reach each of them. According to the model in Section 5.2.1, the source knowledge is $\mathcal{K}_s = (S \times A \times S \times R)^{N_s}$, and for any specific realization $K \in \mathcal{K}_s$, the transfer algorithm returns $\mathcal{A}_{\text{transfer}}(K) = (O, \mathcal{H})$, where $O = \{o_i\}_{i=1}^d$ and $\mathcal{H} = \{h : S \times \{A \cup O\} \rightarrow \mathbb{R}\}$. The learning algorithm can now use the new augmented action space to learn the solution to the target task using option Q-learning (Sutton et al, 1999). Although all these transfer algorithms share the same structure, the critical point is how to identify the subgoals and learn

options from the estimated dynamics. McGovern and Barto (2001) define the concept of bottleneck state as a state which is often traversed by the optimal policy of the source task and that can be considered as critical to solve tasks in the same MDP. Metrics defined for graph partitioning techniques are used in (Menache et al, 2002) and (Simsek et al, 2005) to identify states connecting different regions of the state space. Hengst (2003) proposes a method to automatically develop a MAXQ hierarchy on the basis of the concept of access states. Finally, Bonarini et al (2006) a psychology-inspired notion of interest aimed at identifying states from which the environment can be easily explored.

Action space transfer. A different approach to representation-transfer involving the action space is proposed in (Sherstov and Stone, 2005). Using random task perturbation, a set of tasks is artificially generated from one single source task and a new action set is obtained by removing from A all the actions which are not optimal in any of the source tasks. In this case, the transfer algorithm returns a pair (A', \mathcal{H}) where $A' \subseteq A$ is a subset of the original action space A and $\mathcal{H} = \{h : S \times A' \rightarrow \mathbb{R}\}$. With a smaller action set the learning speed in the target task is significantly improved at the cost of a loss in the optimality of the learned policy. In fact, some of the actions removed according to the artificially generated source tasks might be optimal in the target task. Nonetheless, if source and target tasks are similar and the perturbed tasks are different enough from the source task, then the method is likely to preserve most of the actions necessary to solve the target task.

Feature discovery. In (Mahadevan and Maggioni, 2007; Ferguson and Mahadevan, 2006; Ferrante et al, 2008) a representation-transfer approach similar to option-transfer is proposed. The main difference is that instead of options, the transfer algorithm extracts a set of features $\{\varphi_i\}_{i=1}^d$ which defines the hypothesis space \mathcal{H} . Similar to the option-transfer algorithms, the tasks are assumed to share the same dynamics and an estimated transition model \hat{T}_s is used to extract the features. While the source knowledge \mathcal{K}_s is again the set of samples collected from M_s , the transferred knowledge is $\mathcal{K}_{\text{transfer}} = \mathcal{F}^d$ and once a specific set of features $\{\varphi_i\}_{i=1}^d$ (with $\varphi_i \in \mathcal{F}$) is extracted, the solution space is defined as $\mathcal{H} = \{h(x, a) = \sum_{i=1}^d \alpha_i \varphi_i(x, a)\}$. Furthermore, in option-transfer the objective is to improve the learning speed, while feature-transfer aims at achieving a better approximation of the target value function (i.e., asymptotic improvement). While option-transfer approaches are specifically designed for on-line algorithms such as option Q-learning, the feature-transfer algorithms can be paired to any RL algorithm using a linear function approximation scheme. Mahadevan and Maggioni (2007) introduces a method to generate *proto-value functions* (i.e., the features) using spectral analysis of the Laplacian of the estimated graph of the source MDP. Proto-value functions capture the intrinsic structure of the manifold underlying the dynamics of the tasks at hand (e.g., symmetries) and thus they are likely to approximate well the value function of any task sharing the same dynamics. Ferguson and Mahadevan (2006) generalize this approach to problems with slightly different dynamics. Finally, Ferrante et al (2008) further generalize this approach to a more general setting in which both the

dynamics and reward function may be different in source and target task. A different method is proposed to build the source graph and extract proto-value functions which are well suited to approximate functions obtained from similar dynamics and reward functions.

5.3.3 Parameter Transfer

All the previous methods about representation transfer rely on the implicit assumption that source and target tasks are *similar* enough so that options or features extracted from the source task are effective in learning the solution of the target task. Nonetheless, it is clear that many different notions of similarity can be defined. For instance, we expect the option-transfer methods to work well whenever the two optimal policies have some parts in common (e.g., they both need passing through some specific states to achieve the goal), while proto-value functions are effective when the value functions preserve the structure of the transition graph (e.g., symmetries). The only explicit attempt to measure the expected performance of transfer from source to target as a function of a distance between the two MDPs is pursued by Ferns et al (2004) and Phillips (2006). In particular, they analyze the case in which a policy π_s is transferred from source to target task. The learning process in the target task is then initialized using π_s and its performance is measured. If the MDPs are similar enough, then we expect this policy-transfer method to achieve a jumpstart improvement. According to the formalism introduced in Section 5.2.1, in this case \mathcal{K}_s is any knowledge collected from the source task used to learn π_s , while the transferred knowledge $\mathcal{K}_{\text{transfer}}$ only contains π_s and no learning phase actually takes place. Phillips (2006) defines a state distance between M_s and M_t along the lines of the metrics proposed in (Ferns et al, 2004). In particular, the distance $d : S \rightarrow \mathbb{R}$ is defined as

$$d(s) = \max_{a \in A} (|R_s(s, a) - R_t(s, a)| + \gamma \mathcal{T}(d)(T_s(\cdot | s, a), T_t(\cdot | s, a))), \quad (5.6)$$

where $\mathcal{T}(d)$ is the Kantorovich distance which measures the difference between the two transition distributions $T_s(\cdot | s, a)$ and $T_t(\cdot | s, a)$ given the state distance d . The recursive Equation (5.6) is proved to have a fixed point d^* which is used a state distance. Phillips (2006) prove that when a policy π_s is transferred from source to target, its performance loss w.r.t. the optimal target policy π_t^* can be upper bounded by d^* as

$$\|V_t^{\pi_s} - V_t^{\pi_t^*}\| \leq \frac{2}{1-\gamma} \max_{s \in S} d^*(s) + \frac{1+\gamma}{1-\gamma} \|V_s^{\pi_s} - V_s^{\pi_s^*}\|.$$

As it can be noticed, when the transferred policy is the optimal policy π_s^* of the source task, then its performance loss is upper bounded by the largest value of d^* which takes into consideration the difference between the reward functions and transition models of the two tasks at hand.

As discussed in Section 5.5, many other approaches parameter-transfer approaches have been investigated in the setting of source and target tasks with different state-action spaces.

5.4 Methods for Transfer across Tasks with a Fixed State-Action Space

While in the previous section we considered the setting in which only one source task is available, here we review the main transfer approaches to the general setting when a set of source tasks is available. Transfer algorithms in this setting should deal with two main issues: how to merge knowledge coming from different sources and how to avoid the transfer from sources which differ too much from the target task (*negative* transfer).

5.4.1 Problem Formulation

In this section we consider the more general setting in which the environment \mathcal{E} is defined by a set of tasks \mathcal{M} and a distribution Ω . Similar to the setting in Section 5.3.1, here all the tasks share the same state-action space, that is for any $M \in \mathcal{M}$, $S_M = S$ and $A_M = A$. Although not all the approaches reviewed in the next section explicitly define a distribution Ω , they all rely on the implicit assumption that all the tasks involved in the transfer problem share some characteristics in the dynamics and reward function and that by observing a number of source tasks, the transfer algorithm is able to generalize well across all the tasks in \mathcal{M} .

Example 4. Let us consider a similar scenario to the real-time strategy (RTS) game introduced in (Mehta et al, 2008). In RTS, there is a number of basic tasks such as attacking the enemy, mining gold, building structures, which are useful to accomplish more complex tasks such as preparing an army and conquering an enemy region. The more complex tasks can be often seen as a combination of the low level tasks and the specific combination depends also on the phase of the game, the characteristics of the map, and many other parameters. A simple way to formalize the problem is to consider the case in which all the tasks in \mathcal{M} share the same state-action space and dynamics but have different rewards. In particular, each reward function is the result of a linear combination of a set of d basis reward function, that is, for each task M , the reward is defined as $R_M(\cdot) = \sum_{i=1}^d w_i r_i(\cdot)$ where w is a weight vector and $r_i(\cdot)$ is a basis reward function. Each basis reward function encodes a specific objective (e.g., *defeat the enemy, collect gold*), while the weights represent a combination of them as in a multi-objective problem. It is reasonable to assume that the specific task at hand is randomly generated by setting the weight vector w .

In particular, we can define a generative distribution Ω_ψ with hyper-parameters ψ from which the weights w_M are drawn. For instance, the hyper-parameter could be a pair $\psi = (\mu, \Sigma)$ and Ω_ψ could be a multivariate d -dimensional Gaussian distribution $\mathcal{N}(\mu, \Sigma)$. In this case, the objective of the transfer algorithm is to estimate as accurately as possible the parameters Ψ so as to have a reliable prior on any new weight vector w_M . \square

5.4.2 Instance Transfer

The main idea of instance-transfer algorithms is that the transfer of source samples may improve the learning on the target task. Nonetheless, if samples are transferred from sources which differ too much from the target task, then *negative* transfer might occur. In this section we review the only instance-transfer approach for this transfer setting proposed in (Lazaric et al, 2008) which selectively transfers samples on the basis of the similarity between source and target tasks.

Let L be the number of source tasks, Lazaric et al (2008) propose an algorithm proposed which first collects N_s samples for each source task $\mathcal{K}_s = (S \times A \times S \times R)^{N_s}$ and N_t samples (with $N_t \ll N_s$) from the target task $\mathcal{K}_t = (S \times A \times S \times R)^{N_t}$, and the transfer algorithm takes as input \mathcal{K}_s and \mathcal{K}_t . Instead of returning as output a set containing all the source samples, the method relies on a measure of similarity between the source and the target tasks to select which source samples should be included in $\mathcal{K}_{\text{transfer}}$. Let $K_{s_l} \in \mathcal{K}_s$ and $K_t \in \mathcal{K}_t$ be the specific source and target samples available to the transfer algorithm. The number of source samples is assumed to be large enough to build an accurate kernel-based estimation of each source model \hat{M}_{s_l} . Given the estimated model, the similarity between the source task M_{s_l} and the target task M_t is defined as

$$\Lambda_{s_l} = \frac{1}{N_t} \sum_{n=1}^{N_t} \mathbb{P}(\langle s_n, a_n, s'_n, r_n \rangle | \hat{M}_{s_l})$$

where $\mathbb{P}(\langle s_n, a_n, s'_n, r_n \rangle | \hat{M}_{s_l})$ is the probability of the transition $\langle s_n, a_n, s'_n, r_n \rangle \in K_t$ according to the (estimated) model of M_{s_l} . The intuition behind this measure of similarity is that it is more convenient to transfer samples collected from source tasks which are likely to generate target samples. Finally, source samples are transferred proportionally to their similarity Λ_{s_l} to the target task. The method is further refined using another measure of utility for each source sample so that from each source task only the samples that are more likely to improve the learning performance in the target task. In the experiments reported in (Lazaric et al, 2008) this method is shown to successfully identify which sources are more relevant to transfer samples from and to avoid negative transfer.

5.4.3 Representation Transfer

Unlike in the source-to-target transfer, the objective of representation transfer in this setting is to infer from the source tasks general characteristics that are preserved across the tasks in \mathcal{M} and that can be effectively exploited to improve the average (measured according to the distribution Ω) learning performance w.r.t. single-task learning.

Option transfer. Bernstein (1999) introduce a new option (called *reuse* option) which is added to the set of available actions and which is built as a combination of the optimal policies learned on the L source tasks and it is then reused to speed-up learning on the target task. The process can be re-iterated so that the target task is added to the source tasks and the reuse option is updated accordingly. After a sufficient number of source tasks, the reuse option is shown to significantly speed-up the learning process on new tasks. Options are used to speed-up learning also in (Perkins and Precup, 1999) where a POMDP-like framework is considered. In particular, they assume the set \mathcal{M} to be known in advance and a belief about the identity of the current task is estimated. The value of an option in the current task is then computed as an average of its value in different tasks weighted by the current belief. Although in both these approaches the tasks are assumed to be drawn from a common distribution Ω , they do not provide any analysis about which options could guarantee the best improvement. Kalmar and Szepesvari (1999) consider the problem of finding the set of options which reduces the most the number of iterations needed for value iteration to converge. Unlike the previous approaches, in this case the transfer algorithm is guaranteed to return the optimal set of options. Finally, Asadi and Huber (2007) propose a method for incremental discovery of skills using the MAX-Q hierarchical architecture.

Feature transfer to speed-up learning. A different approach to representation-transfer is to identify a function space which is likely to contain functions able either to speed-up the learning process or to accurately approximate the optimal value functions in \mathcal{M} . Similar to option-transfer, the first objective is usually achieved in the setting of discrete MDPs in which a tabular approach is used. In this case, the space of functions $\mathcal{H} = \{h : S \times A \rightarrow \mathbb{R}\}$ already guarantees the possibility to compute the optimal action value function with an arbitrary small approximation error. Nonetheless, when the number of states and actions is large, the learning process could be very slow. The augmentation of the space \mathcal{H} with features which accurately approximate the optimal action-value functions of the tasks in \mathcal{M} in some parts of the state-action space could considerably speed-up the learning process. The transfer phase in this case takes as input for each source task a source knowledge $K_{s_l} \in \mathcal{K}_{s_l}$ defined as $K_{s_l} = \left\{ \{\langle s_n, a_n, s'_n, r_n \rangle\}_{n=1}^{N_s}, \mathcal{H} \right\}$ while no knowledge about the target task is available. The output knowledge is a set of d new features $\mathcal{H}_{\text{transfer}} = \mathcal{F}^d$, so that $\mathcal{A}_{\text{transfer}}(\{K_{s_l}\}_{l=1}^L) = \{\varphi_i \in \mathcal{F}, i = 1, \dots, d\}$ where the new features are used by the learning algorithm in addition to \mathcal{H} . Foster and Dayan (2002) propose an automatic method to decompose the MDPs into elemental

fragments. In particular, an unsupervised method is first used to analyze the optimal value functions learned so far and to decompose the state space into fragments. Each fragment (i.e., a sub-task) can be solved independently and its value function is then used as an additional feature when learning the target task. Drummond (2002) propose a similar method which identifies subtasks and features according to the analysis of the dynamics of the source tasks. In both cases, the methods are able to identify useful features in maze problems with highly structured optimal value functions. Madden and Howley (2004) introduce a hybrid representation-parameter transfer approach. According to the Q-tables learned on the source tasks, a symbolic learner generates a set of decision rules defined on a higher-level of abstraction compared to the state features used in learning the Q-tables. This representation is then transferred to the target task together with the rules which are used to initialize the Q-table for the target task.

Feature transfer to improve asymptotic performance. While the previous methods consider discrete MDPs where it is possible to exactly compute the optimal solution and the objective is to speed-up the learning, other methods focus on the continuous state-action spaces in which function approximation is mandatory. Walsh et al (2006) consider a simple state-aggregation function approximation scheme. In this case, the objective is to find an aggregation of states able to accurately approximate all the optimal value functions of the tasks at hand. A similar objective is pursued in (Lazaric, 2008) in which the transfer algorithm identifies the best set of features to approximate the source tasks and then reuses them in solving the target task. Similar to (Argyriou et al, 2008), the algorithm relies on the assumption that there exists a small subset of features which is useful to approximate the value functions of the tasks in \mathcal{M} (*shared sparsity assumption*). The algorithm considers a set of features $\{\varphi_i^\theta\}_{i=1}^d$ parameterized by a parameter $\theta \in \Theta$ and the corresponding linear hypothesis space $\mathcal{H}_\theta = \{h(x,a) = \sum_{i=1}^d \alpha_i \varphi_i^\theta(x,a)\}$. The objective is to learn the parameter θ which defines a feature space such that only a small set of features are used to approximate the value functions of the tasks (i.e., features such that the corresponding optimal weights α have a small number of non-zero components). Empirical evaluation shows that whenever the optimal value functions of the tasks in \mathcal{M} can be represented with a very small number of features, the algorithm is able to learn them and to obtain a significant convergence improvement.

5.4.4 Parameter Transfer

Unlike representation-transfer approaches, all parameter-transfer algorithms explicitly define a distribution Ω on the task space \mathcal{M} and try to estimate the true distribution in order to build a prior over the space of hypotheses \mathcal{H} so as to improve the initial performance (jumpstart improvement) and reduce the number of samples needed to solve any task in Ω . More formally, most of the parameter-transfer approaches share the following structure. Let $\mathcal{M} = \{M_\theta, \theta \in \Theta\}$ be a parameterized space of tasks with parameters θ and Ω_ψ a family of task probability distributions,

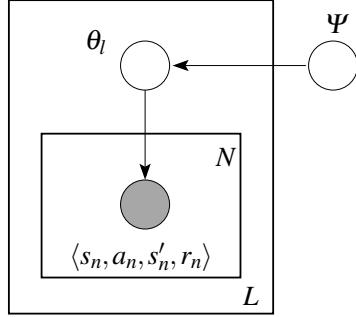


Fig. 5.5 Example of a generative model of a hierarchical Bayesian model (HBM). The observations $\langle s, a, s', r \rangle$ are generated according to an MDP parameterized by a parameter vector θ , while each task is generated according to a distribution defined by a set of hyper-parameters Ψ . Ψ_0 is a vector of parameters defining the prior over the hyper-parameters.

where $\psi \in \Psi$ is a hyper-parameter vector. The main assumption is that all the task parameters θ are independently and identically distributed according to a specific task distribution Ω_{ψ^*} .

The hierarchical Bayesian model. The structure of this problem is usually represented as a hierarchical Bayesian model (HBM) as depicted in Figure 5.5. The transfer algorithms take as input samples $K_{s_l} = \left\{ \langle s_n, a_n, s'_n, r_n \rangle \right\}_{n=1}^{N_s}$ from each of the source tasks M_{θ_l} ($l = 1, \dots, L$) which are drawn from the true distribution Ω_{ψ^*} (i.e., $\theta_l \stackrel{iid}{\sim} \Omega_{\psi^*}$) whose true hyper-parameters are unknown. Given a prior over ψ , the algorithm solves the inference problem

$$\mathbb{P}(\psi | \{K_{s_l}\}_{l=1}^L) \propto \prod_{l=1}^L \mathbb{P}(K_{s_l} | \psi) \mathbb{P}(\psi), \quad (5.7)$$

where $\mathbb{P}(K_{s_l} | \psi) \propto \mathbb{P}(K_{s_l} | \theta) \mathbb{P}(\psi)$. The ψ with highest probability is usually transferred and used to initialize the learning process on the target task. Notice that the learning algorithm must be designed so as to take advantage of the knowledge about the specific task distribution Ω_ψ returned by the transfer phase. Bayesian algorithms for RL such as GPTD (Engel et al, 2005) are usually adopted (see Chapter 11).

The inference problem in Equation (5.7) leverages on the knowledge collected on all the tasks at the same time. Thus, even if few samples per task are available (i.e., N_s is small), the algorithm can still take advantage of a large number of tasks (i.e., L is large) to solve the inference problem and learn an accurate estimate of ψ^* . As L increases the hyper-parameter ψ gets closer and closer to the true hyper-parameter ψ^* and ψ can be used to build a prior on the parameter θ for any new target task drawn from the distribution Ω_{ψ^*} . Depending on the specific definition of Θ and Ω_ψ and the way the inference problem is solved, many different algorithms can be deduced from this general model.

Inference for transfer. Tanaka and Yamamura (2003) consider a simpler approach. Although the MDPs are assumed to be drawn from a distribution Ω , the proposed algorithm does not try to estimate the task distribution but only a statistics about the action values is computed. The mean and variance of the action values over different tasks are computed and then used to initialize the Q-table for new tasks. Sunmola and Wyatt (2006) and Wilson et al (2007) consider the case where the MDP dynamics and reward function are parameterized by a parameter vector θ and they are assumed to be drawn from a common distribution Ω_ψ . The inference problem is solved by choosing appropriate conjugate priors over the hyper-parameter ψ . A transfer problem on POMDPs is considered in (Li et al, 2009). In this case, no explicit parameterization of the tasks is provided. On the other hand, it is the space of history-based policies \mathcal{H} which is parameterized by a vector parameter $\theta \in \Theta$. A Dirichlet process is then used as a non-parametric prior over the parameters of the optimal policies for different tasks. Lazaric and Ghavamzadeh (2010) consider the case of a parameterized space of value functions by considering the space of linear functions spanned by a given set of features, $\mathcal{H} = \{h(x,a) = \sum_{i=1}^d \theta_i \varphi_i(x,a)\}$. The vector θ is assumed to be drawn from a multivariate Gaussian with parameters ψ drawn from a normal-inverse-Wishart hyper-prior (i.e., $\theta \sim \mathcal{N}(\mu, \Sigma)$ and $\mu, \Sigma \sim \mathcal{N}\text{-}\mathcal{IW}(\psi)$). The inference problem is solved using an EM-like algorithm which takes advantage of the conjugate priors. This approach is further extended to consider the case in which not all the tasks are drawn from the same distribution. In order to cluster tasks into different classes, Lazaric and Ghavamzadeh (2010) place a Dirichlet process on the top of the hierarchical Bayesian model and the number of classes and assignment of tasks to classes is automatically learned by solving an inference problem using a Gibbs sampling method. Finally, Mehta et al (2008) define the reward function as a linear combination of reward features which are common across tasks, while the weights are specific for each task. The weights are drawn from a distribution Ω and the transfer algorithm compactly stores the optimal value functions of the source tasks exploiting the structure of the reward function and uses them to initialize the solution in the target task.

5.5 Methods for Transfer from Source to Target Tasks with a Different State-Action Spaces

All the previous settings consider the case where all the tasks share the same *domain* (i.e., they have the same state-action space). In the most general transfer setting the tasks in \mathcal{M} may also differ in terms of number or range of the state-action variables.

5.5.1 Problem Formulation

Although here we consider the general case in which each task $M \in \mathcal{M}$ is defined as an MDP $\langle S_M, A_M, T_M, R_M \rangle$ and the environment \mathcal{E} is obtained by defining a

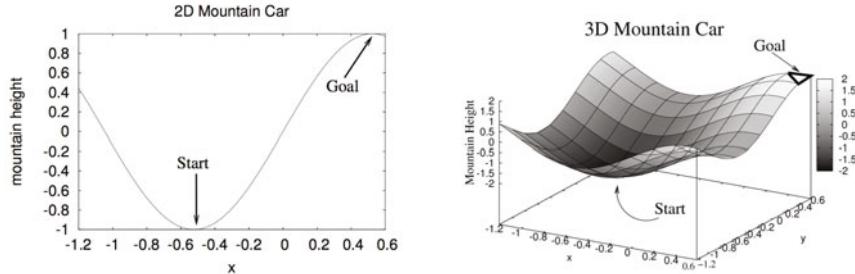


Fig. 5.6 Transfer from 2D to 3D mountain car (Taylor et al, 2008a)

distribution Ω on \mathcal{M} , in practice only the source-to-target transfer setting has been considered. Thus, similarly to Section 5.3, we consider a task space $\mathcal{M} = \{M_s, M_t\}$, where $M_s = \langle S_s, A_s, T_s, R_s \rangle$ is the source task and $M_t = \langle S_t, A_t, T_t, R_t \rangle$ is the target task on which we want to improve the learning performance. According to the notation introduced in Section 5.2.1, \mathcal{H}_s and \mathcal{H}_t are now defined on different state-action spaces and the knowledge in \mathcal{H}_s cannot be directly used by the learning algorithm to learn on the target task. Thus, the transfer phase implemented by $\mathcal{A}_{\text{transfer}}$ must return some knowledge $\mathcal{H}_{\text{transfer}}$ compatible with \mathcal{H}_t . This objective is usually achieved following three different approaches: **transform \mathcal{H}_s into \mathcal{H}_t through a hand-coded mapping, learn a mapping from M_s to M_t and transform \mathcal{H}_s into \mathcal{H}_t , extract from \mathcal{H}_s some abstract knowledge that can be reused to solve M_t .** In the next section we still follow the different categories used in the previous sections but we will make explicit which approach to the problem of mapping is used.

Example 5. Let us consider the mountain car domain and the source and target tasks in Figure 5.6 introduced by Taylor et al (2008a). Although the problem is somehow similar (i.e., an under-powered car has to move from the bottom of the valley to the top of the hill), the two tasks are defined over a different state space and the action space contains a different number of actions. In fact, in the 2D mountain car task the state space is defined by the position and the velocity variables (x, \dot{x}) and the action space contains the actions $A = \{\text{Left}, \text{Neutral}, \text{Right}\}$. On the other hand, the 3D task has two additional state variables describing the position in y and its corresponding speed \dot{y} and the action space becomes $A = \{\text{Neutral}, \text{West}, \text{East}, \text{South}, \text{North}\}$. The transfer approaches described so far cannot be applied here because the knowledge $\mathcal{H}_{\text{transfer}}$ they transfer from the source task would not be compatible with the target task. In this case, the transfer algorithm must define a suitable mapping between source and target state and action spaces, and then transfer solutions learned in the 2D mountain car to initialize the learning process in the 3D task. \square

5.5.2 Instance Transfer

Similar to the method introduced by Lazaric et al (2008), Taylor et al (2008a) study the transfer of samples. Unlike Lazaric et al (2008), here only one source task is considered and no specific method for the selection of the samples is implemented. On the other hand, the two tasks do not share the same domain anymore, thus an explicit mapping between their state-action variables is needed. A hand-coded mapping is provided as input to the transfer algorithms which simply applies it to the source samples thus obtaining samples that can be used to learn on the target task. Following the inter-task mapping formalism introduced by Taylor et al (2007a) and used by Taylor et al (2008a), a hand-coded mapping is defined by two functionals χ_S and χ_A . Let the state spaces S_s and S_t be factorized in d_s and d_t state variables respectively (i.e., $S_s = S^1 \times \dots \times S^{d_s}$ and $S_t = S^1 \times \dots \times S^{d_t}$) and A_s and A_t be scalar spaces with values $A_s = \{a^1, \dots, a^{k_s}\}$ and $A_t = \{a^1, \dots, a^{k_t}\}$. The state mapping maps the index of a source state variable $1 \leq i \leq d_s$ to the index of a state variable $1 \leq j \leq d_t$ in the target state space, that is $\chi_S : \{1, \dots, d_s\} \rightarrow \{1, \dots, d_t\}$. With an abuse of notation we denote by $\chi_S(s) \in S_t$ the transformation of a state $s \in S_s$ into the state obtained by mapping each variable of s into a target variable according to χ_S . Similarly, the action mapping χ_A maps each source action in A_s to one of the target actions in A_t . As a result, if \mathcal{H}_s is the space of source samples and $K_s \in \mathcal{H}_s$ is one specific realization of N_s samples, for any samples $\langle s_n, a_n, s'_n, r_n \rangle \in K_s$ the transfer algorithm returns new target samples $\langle \chi_S(s_n), \chi_A(a_n), \chi_S(s'_n), r_n \rangle$. While Lazaric et al (2008) define an algorithm where the transferred samples are used in a batch model-free RL algorithm, Taylor et al (2008a) study how the model-based algorithm Fitted R-max can benefit from samples coming from the source task when transformed according to a hand-coded mapping from the source to the target task. In particular, the method is shown to be effective in the generalized mountain car problem in Figure 5.6.

5.5.3 Representation Transfer

As reviewed in the previous sections, many transfer approaches develop options that can be effectively reused in the target task. In this case, the main problem is that options learned on the source task are defined as a mapping from S_s to A_s and they cannot be used in a target task with different state-action variables. A number of transfer algorithms deal with this problem by considering abstract options that can be reused in different tasks. Ravindran and Barto (2003); Soni and Singh (2006) use the *homomorphism* framework to map tasks to a common abstract level. For instance, let us consider all the navigation problems in an empty squared room. In this case, it is possible to define one common abstract MDP and obtain any specific MDP by simply using operators such as translation, scaling, and rotation. In order to deal with this scenario, Ravindran and Barto (2003) introduce the concept of relativized options. Unlike traditional options, relativized options are defined on the abstract MDP, without an absolute frame of reference, and their policy is then transformed according to the specific target task at hand. In particular, a set of possible

transformations is provided and the transfer phase needs to identify the most suitable transformation of the relativized options depending on the current target task. The problem is casted as a Bayesian parameter estimation problem and the transformation which makes the sequence of states observed by following the option more likely is selected. Konidaris and Barto (2007) define options at a higher level of abstraction and they can be used in the target task without any explicit mapping or transformation. In fact, portable options are defined in a non-Markovian *agent space* which depends on the characteristics of the agent and remains fixed across tasks. This way, even when tasks are defined on different state-action spaces, portable options can be reused to speed-up learning in any target task at hand. Finally, Torrey et al (2006) proposed an algorithm in which a set of skills is first identified using inductive logic programming and then reused in the target task by using a hand-coded mapping from source to target.

5.5.4 Parameter Transfer

While in the setting considered in Section 5.3, the transfer of initial solutions (e.g., the optimal source policy) from the source to the target task is trivial, in this case the crucial aspect in making transfer effective is to find a suitable mapping from the source state-action space $S_s \times A_s$ to the target state-action space $S_t \times A_t$.

Most of the algorithms reviewed in the following consider hand-coded mappings and investigate how the transfer of different sources of knowledge (e.g., policies, value functions) influence the performance on the target task. The transformation through a hand-coded mapping and the transfer of the source value function to initialize the learning in the target task has been first introduced by Taylor and Stone (2005) and Taylor et al (2005) and its impact has been study in a number of challenging problems such as the simulated keep-away problem (Stone et al, 2005). Banerjee and Stone (2007) also consider the transfer of value functions in the context of general games where different games can be represented by a common abstract structure. Torrey et al (2005) learn the Q-table in the target task by reusing *advices* (i.e., actions with higher Q-values in the source task) which are mapped to the target task through a hand-coded mapping. While the previous approaches assume that in both source and target task the same solution representation is used (e.g., a tabular approach), Taylor and Stone (2007) consider the problem of mapping a solution (i.e., a value function or a policy) to another solution when either the approximation architecture (e.g., CMAC and neural networks) or the learning algorithm itself (e.g., value-based and policy search methods) changes between source and target tasks. Using similar mappings as for the state-action mapping, they show that transfer is still possible and it is still beneficial in improving the performance on the target task. Finally, Taylor et al (2007b) study the transfer of the source policy where a hand-coded mapping is used to transform the source policy into a valid policy for the target task and a policy search algorithm is then used to refine it.

Unlike the previous approaches, some transfer methods automatically identify the most suitable mapping between source and target tasks. Taylor et al (2008b) introduce the MASTER algorithm. The objective is to identify among all the possible mappings from source to target state variables, the one which guarantees the best prediction of the dynamics of the environment. The algorithm receives as input a relatively large number of samples from the source task and few target samples. Let X be the set of all the possible mappings between S_s and S_t , $K_s \in \mathcal{K}_s$ and $K_t \in \mathcal{K}_t$ be specific realization of source and target samples. From K_t the algorithm first computes an approximation of the target dynamics \hat{T}_t . Each sample $\langle s_n, a_n, s'_n, r_n \rangle$ in K_s is then transformed with one of the possible mappings $\chi_s \in X$ and the state $\chi_s(s'_n)$ is compared to the state predicted by $\hat{T}_t(\chi_s(s_n), a)$. The mapping χ_s which is the most accurate in predicting the transitions of the samples in K_s is then used to transfer the solution of the source task. Talvitie and Singh (2007) first learn a policy for the source task, and a target policy is then obtained by mapping the source policy according to each of the possible mappings from the source to the target state variables. The problem of selecting the most appropriate mapping is then translated into the problem of evaluating the best policy among the target policies. The problem is then solved using an expert-like algorithm (Cesa-Bianchi and Lugosi, 2006).

5.6 Conclusions and Open Questions

In this chapter we defined a general framework for the transfer learning problem in the reinforcement learning paradigm, we proposed a classification of the different approaches to the problem, and we reviewed the main algorithms available in the literature. Although many algorithms have been already proposed, the problem of transfer in RL is far from being solved. In the following we single out a few open questions that are relevant to the advancement of the research on this topic. We refer the reader to the survey by Taylor and Stone (2009) for other possible lines of research in transfer in RL.

Theoretical analysis of transfer algorithms. Although experimental results support the idea that RL algorithms can benefit from transfer from related tasks, no transfer algorithm for RL has strong theoretical guarantees. Recent research in transfer and multi-task learning in the supervised learning paradigm achieved interesting theoretical results identifying the conditions under which transfer approaches are expected to improve the performance over single-task learning. Crammer et al (2008) study the performance of learning reusing samples coming from different classification tasks and they prove that when the sample distributions of the source tasks do not differ too much compared to the target distribution, then the transfer approach performs better than just using the target samples. Baxter (2000) studies the problem of learning the most suitable set of hypotheses for a given set of tasks. In particular, he shows that, as the number of source tasks increases, the transfer algorithm manages to identify a hypothesis set which is likely to contain *good* hypotheses for all the tasks in \mathcal{M} . Ben-David and Schuller-Borbely (2008) consider the problem of

learning the best hypothesis set in the context of multi-task learning where the objective is not to generalize on new tasks but to achieve a better average performance in the source tasks. At the same time, novel theoretical results are now available for a number of popular RL algorithms such as fitted value iteration (Munos and Szepesvári, 2008), LSTD (Farahmand et al, 2008; Lazaric et al, 2010), and Bellman residual minimization (Antos et al, 2008; Maillard et al, 2010). An interesting line of research is to take advantage of theoretical results of transfer algorithms in the supervised learning setting and of RL algorithms in the single-task case to develop new RL transfer algorithms which provably improve the performance over single-task learning.

Transfer learning for exploration. The objective of learning speed improvement (see Section 5.2.2) is often achieved by a better use of the samples at hand (e.g., by changing the hypothesis set) rather than by the collection of more *informative* samples. This problem is strictly related to the exploration-exploitation dilemma where the objective is to trade-off between the exploration of different strategies and the exploitation of the best strategy so far. Recent works by Bartlett and Tewari (2009); Jaksch et al (2010) studied optimal exploration strategies for single-task learning. Although most of the option-based transfer methods implicitly bias the exploration strategy, the problem of how the exploration on one task should be adapted on the basis of the knowledge of previous related tasks is a problem which received little attention so far.

Concept drift and continual learning. One of the main assumptions of transfer learning is that a clear distinction between the tasks in \mathcal{M} is possible. Nonetheless, in many interesting applications there is no sharp division between source and target tasks while it is rather the task itself that changes in time. This problem, also known as *concept drift*, is also strictly related to the continual learning and lifelong learning paradigm (Silver and Poirier, 2007) in which, as the learning agent autonomously discovers new regions of a non-stationary environment, it also increases its capability to solve tasks defined on that environment. Although tools coming from transfer learning probably could be reused also in this setting, novel approaches are needed to deal with the non-stationarity of the environment and to track the changes in the task at hand.

References

- Antos, A., Szepesvári, C., Munos, R.: Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning Journal* 71, 89–129 (2008)
- Argyriou, A., Evgeniou, T., Pontil, M.: Convex multi-task feature learning. *Machine Learning Journal* 73(3), 243–272 (2008)
- Asadi, M., Huber, M.: Effective control knowledge transfer through learning skill and representation hierarchies. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)*, pp. 2054–2059 (2007)

- Banerjee, B., Stone, P.: General game learning using knowledge transfer. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007), pp. 672–677 (2007)
- Bartlett, P.L., Tewari, A.: Regal: a regularization based algorithm for reinforcement learning in weakly communicating mdps. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-2009), pp. 35–42. AUAI Press, Arlington (2009)
- Baxter, J.: A model of inductive bias learning. *Journal of Artificial Intelligence Research* 12, 149–198 (2000)
- Ben-David, S., Schuller-Borbely, R.: A notion of task relatedness yielding provable multiple-task learning guarantees. *Machine Learning Journal* 73(3), 273–287 (2008)
- Bernstein, D.S.: Reusing old policies to accelerate learning on new mdps. Tech. rep., University of Massachusetts, Amherst, MA, USA (1999)
- Bonarini, A., Lazaric, A., Restelli, M.: Incremental Skill Acquisition for Self-motivated Learning Animats. In: Nolfi, S., Baldassarre, G., Calabretta, R., Hallam, J.C.T., Marocco, D., Meyer, J.-A., Miglino, O., Parisi, D. (eds.) SAB 2006. LNCS (LNAI), vol. 4095, pp. 357–368. Springer, Heidelberg (2006)
- Cesa-Bianchi, N., Lugosi, G.: Prediction, Learning, and Games. Cambridge University Press (2006)
- Crammer, K., Kearns, M., Wortman, J.: Learning from multiple sources. *Journal of Machine Learning Research* 9, 1757–1774 (2008)
- Drummond, C.: Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research* 16, 59–104 (2002)
- Engel, Y., Mannor, S., Meir, R.: Reinforcement learning with Gaussian processes. In: Proceedings of the 22nd International Conference on Machine Learning (ICML-2005), pp. 201–208 (2005)
- Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6, 503–556 (2005)
- Farahmand, A.M., Ghavamzadeh, M., Szepesvári, C., Mannor, S.: Regularized policy iteration. In: Proceedings of the Twenty-Second Annual Conference on Advances in Neural Information Processing Systems (NIPS-2008), pp. 441–448 (2008)
- Fawcett, T., Callan, J., Matheus, C., Michalski, R., Pazzani, M., Rendell, L., Sutton, R. (eds.): Constructive Induction Workshop at the Eleventh International Conference on Machine Learning (1994)
- Ferguson, K., Mahadevan, S.: Proto-transfer learning in markov decision processes using spectral methods. In: Workshop on Structural Knowledge Transfer for Machine Learning at the Twenty-Third International Conference on Machine Learning (2006)
- Ferns, N., Panangaden, P., Precup, D.: Metrics for finite markov decision processes. In: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI-2004), pp. 162–169 (2004)
- Ferrante, E., Lazaric, A., Restelli, M.: Transfer of task representation in reinforcement learning using policy-based proto-value functions. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2008), pp. 1329–1332 (2008)
- Foster, D.J., Dayan, P.: Structure in the space of value functions. *Machine Learning Journal* 49(2-3), 325–346 (2002)
- Gentner, D., Loewenstein, J., Thompson, L.: Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology* 95(2), 393–408 (2003)
- Gick, M.L., Holyoak, K.J.: Schema induction and analogical transfer. *Cognitive Psychology* 15, 1–38 (1983)

- Hauskrecht, M.: Planning with macro-actions: Effect of initial value function estimate on convergence rate of value iteration. Tech. rep., Department of Computer Science, University of Pittsburgh (1998)
- Hengst, B.: Discovering hierarchy in reinforcement learning. PhD thesis, University of New South Wales (2003)
- Jaksch, T., Ortner, R., Auer, P.: Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research* 11, 1563–1600 (2010)
- Kalmar, Z., Szepesvári, C.: An evaluation criterion for macro-learning and some results. Tech. Rep. TR-99-01, Mindmaker Ltd. (1999)
- Konidaris, G., Barto, A.: Autonomous shaping: knowledge transfer in reinforcement learning. In: Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-2006), pp. 489–496 (2006)
- Konidaris, G., Barto, A.G.: Building portable options: Skill transfer in reinforcement learning. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007), pp. 895–900 (2007)
- Langley, P.: Transfer of knowledge in cognitive systems. In: Talk, Workshop on Structural Knowledge Transfer for Machine Learning at the Twenty-Third International Conference on Machine Learning (2006)
- Lazaric, A.: Knowledge transfer in reinforcement learning. PhD thesis, Politecnico di Milano (2008)
- Lazaric, A., Ghavamzadeh, M.: Bayesian multi-task reinforcement learning. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning, ICML-2010 (2010) (submitted)
- Lazaric, A., Restelli, M., Bonarini, A.: Transfer of samples in batch reinforcement learning. In: Proceedings of the Twenty-Fifth Annual International Conference on Machine Learning (ICML-2008), pp. 544–551 (2008)
- Lazaric, A., Ghavamzadeh, M., Munos, R.: Finite-sample analysis of lstd. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning, ICML-2010 (2010)
- Li, H., Liao, X., Carin, L.: Multi-task reinforcement learning in partially observable stochastic environments. *Journal of Machine Learning Research* 10, 1131–1186 (2009)
- Madden, M.G., Howley, T.: Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review* 21(3-4), 375–398 (2004)
- Mahadevan, S., Maggioni, M.: Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research* 38, 2169–2231 (2007)
- Maillard, O.A., Lazaric, A., Ghavamzadeh, M., Munos, R.: Finite-sample analysis of bellman residual minimization. In: Proceedings of the Second Asian Conference on Machine Learning, ACML-2010 (2010)
- McGovern, A., Barto, A.G.: Automatic discovery of subgoals in reinforcement learning using diverse density. In: Proceedings of the Eighteenth International Conference on Machine Learning, ICML 2001 (2001)
- Mehta, N., Natarajan, S., Tadepalli, P., Fern, A.: Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning Journal* 73(3), 289–312 (2008)
- Menache, I., Mannor, S., Shimkin, N.: Q-cut - dynamic discovery of sub-goals in reinforcement learning. In: Proceedings of the Thirteen European Conference on Machine Learning, pp. 295–306 (2002)
- Munos, R., Szepesvári, C.: Finite time bounds for fitted value iteration. *Journal of Machine Learning Research* 9, 815–857 (2008)

- Pan, S.J., Yang, Q.: A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22(22), 1345–1359 (2010)
- Perkins, D.N., Salomon, G., Press, P.: Transfer of learning. In: *International Encyclopedia of Education*. Pergamon Press (1992)
- Perkins, T.J., Precup, D.: Using options for knowledge transfer in reinforcement learning. Tech. rep., University of Massachusetts, Amherst, MA, USA (1999)
- Phillips, C.: Knowledge transfer in markov decision processes. McGill School of Computer Science (2006), <http://www.cs.mcgill.ca/~martin/usrs/phillips.pdf>
- Ravindran, B., Barto, A.G.: Relativized options: Choosing the right transformation. In: *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*, pp. 608–615 (2003)
- Sherstov, A.A., Stone, P.: Improving action selection in MDP's via knowledge transfer. In: *Proceedings of the Twentieth National Conference on Artificial Intelligence, AAAI-2005* (2005)
- Silver, D.: Selective transfer of neural network task knowledge. PhD thesis, University of Western Ontario (2000)
- Silver, D.L., Poirier, R.: Requirements for Machine Lifelong Learning. In: Mira, J., Álvarez, J.R. (eds.) *IWINAC 2007, Part I. LNCS*, vol. 4527, pp. 313–319. Springer, Heidelberg (2007)
- Simsek, O., Wolfe, A.P., Barto, A.G.: Identifying useful subgoals in reinforcement learning by local graph partitioning. In: *Proceedings of the Twenty-Second International Conference of Machine Learning, ICML 2005* (2005)
- Singh, S., Barto, A., Chentanez, N.: Intrinsically motivated reinforcement learning. In: *Proceedings of the Eighteenth Annual Conference on Neural Information Processing Systems, NIPS-2004* (2004)
- Soni, V., Singh, S.P.: Using homomorphisms to transfer options across continuous reinforcement learning domains. In: *Proceedings of the Twenty-first National Conference on Artificial Intelligence, AAAI-2006* (2006)
- Stone, P., Sutton, R.S., Kuhlmann, G.: Reinforcement learning for RoboCup-soccer keep-away. *Adaptive Behavior* 13(3), 165–188 (2005)
- Sunmola, F.T., Wyatt, J.L.: Model transfer for markov decision tasks via parameter matching. In: *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group, PlanSIG 2006* (2006)
- Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge (1998)
- Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112, 181–211 (1999)
- Talvitie, E., Singh, S.: An experts algorithm for transfer learning. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)*, pp. 1065–1070 (2007)
- Tanaka, F., Yamamura, M.: Multitask reinforcement learning on the distribution of mdps. In: *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, vol. 3, pp. 1108–1113 (2003)
- Taylor, M.E., Stone, P.: Behavior transfer for value-function-based reinforcement learning. In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2005)*, pp. 53–59 (2005)

- Taylor, M.E., Stone, P.: Representation transfer for reinforcement learning. In: AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development (2007)
- Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10(1), 1633–1685 (2009)
- Taylor, M.E., Stone, P., Liu, Y.: Value functions for RL-based behavior transfer: A comparative study. In: Proceedings of the Twentieth National Conference on Artificial Intelligence, AAAI-2005 (2005)
- Taylor, M.E., Stone, P., Liu, Y.: Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research* 8, 2125–2167 (2007a)
- Taylor, M.E., Whiteson, S., Stone, P.: Transfer via inter-task mappings in policy search reinforcement learning. In: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS-2007 (2007b)
- Taylor, M.E., Jong, N.K., Stone, P.: Transferring instances for model-based reinforcement learning. In: Proceedings of the European Conference on Machine Learning (ECML-2008), pp. 488–505 (2008a)
- Taylor, M.E., Kuhlmann, G., Stone, P.: Autonomous transfer for reinforcement learning. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2008), pp. 283–290 (2008b)
- Thorndike, E.L., Woodworth, R.S.: The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review* 8 (1901)
- Torrey, L., Walker, T., Shavlik, J., Maclin, R.: Using Advice to Transfer Knowledge Acquired in one Reinforcement Learning Task to Another. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 412–424. Springer, Heidelberg (2005)
- Torrey, L., Shavlik, J., Walker, T., Maclin, R.: Skill Acquisition Via Transfer Learning and Advice Taking. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 425–436. Springer, Heidelberg (2006)
- Utgoff, P.: Shift of bias for inductive concept learning. *Machine Learning* 2, 163–190 (1986)
- Walsh, T.J., Li, L., Littman, M.L.: Transferring state abstractions between mdps. In: ICML Workshop on Structural Knowledge Transfer for Machine Learning (2006)
- Watkins, C., Dayan, P.: Q-learning. *Machine Learning* 8, 279–292 (1992)
- Wilson, A., Fern, A., Ray, S., Tadepalli, P.: Multi-task reinforcement learning: a hierarchical bayesian approach. In: Proceedings of the Twenty-Forth International Conference on Machine learning (ICML-2007), pp. 1015–1022 (2007)

Chapter 6

Sample Complexity Bounds of Exploration

Lihong Li

Abstract. Efficient exploration is widely recognized as a fundamental challenge inherent in reinforcement learning. Algorithms that explore efficiently converge faster to near-optimal policies. While heuristics techniques are popular in practice, they lack formal guarantees and may not work well in general. This chapter studies algorithms with polynomial sample complexity of exploration, both model-based and model-free ones, in a unified manner. These so-called PAC-MDP algorithms behave near-optimally except in a “small” number of steps with high probability. A new learning model known as KWIK is used to unify most existing model-based PAC-MDP algorithms for various subclasses of Markov decision processes. We also compare the sample-complexity framework to alternatives for formalizing exploration efficiency such as regret minimization and Bayes optimal solutions.

6.1 Introduction

Broadly speaking, an online reinforcement-learning (RL) agent interacts with an initially unknown environment to maximize cumulative rewards by trial and error. Without external supervision, the agent has to experiment with the environment to acquire knowledge based on which its policy is optimized. While the agent strives to maximize its total rewards, it must also try various actions purposely—even if they appear suboptimal—in the hope of collecting information about the environment so as to earn more rewards in the long run. A purely explorative agent that extensively explores the environment is undesirable because such a utility-*blind* agent may collect little reward due to over-exploration. A purely exploiting agent which always picks actions that appear the best is also undesirable as it may end up with a suboptimal action-selection strategy because of its incomplete knowledge of the

Lihong Li
Yahoo! Research, 4401 Great America Parkway, Santa Clara, CA, USA 95054
e-mail: lihong@yahoo-inc.com

environment. The tradeoff between *exploration* and *exploitation* is an intrinsic challenge in essentially all reinforcement-learning problems (Sutton and Barto, 1998).

The necessity of sufficient exploration has been widely acknowledged in the literature for a long time. For example, typical asymptotic convergence guarantees for temporal-difference algorithms require every action be taken in all states infinitely often (Jaakkola et al, 1994; Singh et al, 2000; Watkins and Dayan, 1992). Without infinite exploration, initial errors in the value-function estimate may not be corrected and a temporal difference algorithm may converge to a suboptimal policy.

A number of exploration heuristics have been popular in practice (Sutton and Barto, 1998). Unfortunately, while these strategies are successful in some applications, few guarantees can be made in general. In fact, some of them are *provably inefficient* in certain simple problems, as shown in the next section. Recently, there have been growing interests in formalizing the notion of exploration efficiency as well as developing algorithms with provably efficient exploration. Analysis of this type usually focuses on how fast an RL algorithm's cumulative reward approaches an optimal policy's. Such guarantees are very different from traditional, *asymptotic* ones (Bertsekas and Tsitsiklis, 1996) since they characterize how fast an algorithm can learn to maximize cumulative reward after *finitely* many steps.

The rest of this chapter is organized as follows. Section 6.2 formalizes online reinforcement learning, describes a few popular exploration heuristics, and shows why they may fail by a simple example. Section 6.3 discusses a few ways to formalize exploration efficiency. Due to space limitation, the chapter focuses on one of them known as sample complexity. We also compare and relate it to other possibilities like the Bayesian criterion and regret minimization. A generic theorem is also given in Section 6.4 as the basic theoretical machinery to study the sample complexity of a wide class of algorithms, including model-based approaches in Section 6.5 and model-free ones in Section 6.6. Finally, Section 6.7 concludes this chapter and outlines a few open questions.

6.2 Preliminaries

The chapter studies *online* reinforcement learning, which is more challenging than the offline alternative, and focuses on agents that aim to maximize γ -discounted cumulative reward for some given $\gamma \in (0,1)$.

Throughout the chapter, we model the environment as a Markov decision process (MDP) $M = \langle \mathbf{S}, \mathbf{A}, T, R, \gamma \rangle$ with state space \mathbf{S} , action space \mathbf{A} , transition function T , reward function R , and discount factor γ . Standard notation in reinforcement learning is used, whose definition is found in Chapter 1. In particular, given a policy π that maps \mathbf{S} to \mathbf{A} , $V_M^\pi(s)$ and $Q_M^\pi(s,a)$ are the state and state-action value functions of π in MDP M ; $V_M^*(s)$ and $Q_M^*(s,a)$ are the corresponding optimal value functions. If M is clear from the context, it may be omitted and the value functions are denoted by V^π , Q^π , V^* , and Q^* , respectively. Finally, we use $\tilde{O}(\cdot)$ as a shorthand for

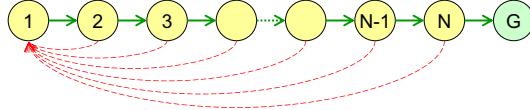


Fig. 6.1 Combination lock (Whitehead, 1991)

the common big-O notation with logarithmic factors suppressed. Online RL is made precise by the following definition.

Definition 6.1. The *online interaction* between the agent and environment, modeled as an MDP $M = \langle S, A, T, R, \gamma \rangle$, proceeds in *discrete timesteps* $t = 1, 2, 3, \dots$,

1. The agent perceives the *current state* $s_t \in S$ of the environment, then takes an action $a_t \in A$.
2. In response, the environment sends an immediate reward $r_t \in [0, 1]$ to the agent,¹ and moves to a *next state* $s_{t+1} \in S$. This *transition* is governed by the dynamics of the MDP. In particular, the expectation of r_t is $R(s_t, a_t)$, and the next state s_{t+1} is drawn randomly from the distribution $T(s_t, a_t, \cdot)$.
3. The clock ticks: $t \leftarrow t + 1$.

In addition, we assume an upper bound V_{\max} on the optimal value function such that $V_{\max} \geq V^*(s)$ for all $s \in S$. Since we have assumed $R(s, a) \in [0, 1]$, we always have $V_{\max} \leq 1/(1 - \gamma)$, although in many situations we have $V_{\max} \ll 1/(1 - \gamma)$.

A number of heuristics have been proposed in the literature. In practice, the most popular exploration strategy is probably ϵ -greedy: the agent chooses a random action with probability ϵ , and the best action according to its current value-function estimate otherwise. While the ϵ -greedy rule often guarantees *sufficient* exploration, it may not be *efficient* since exploration occurs *uniformly* on all actions. A number of alternatives such as soft-max, counter-based, recency-based, optimistic initialization, and exploration bonus are sometimes found more effective empirically (Sutton and Barto, 1998; Thrun, 1992). Advanced techniques exist that guide exploration using interval estimation (Kaelbling, 1993; Meuleau and Bourgine, 1999; Wiering and Schmidhuber, 1998), information gain (Dearden et al, 1999), MDP characteristics (Ratitch and Precup, 2002), or expert demonstration (Abbeel and Ng, 2005), to name a few.

Unfortunately, while these heuristics work well in some applications, they may be inefficient in others. Below is a prominent example (Whitehead, 1991), where ϵ -greedy is provably inefficient.

Example 6.1. Consider an MDP with $N + 1$ states and 2 actions, as depicted in Figure 6.1. State 1 is the start state and state G is the absorbing goal state. Taking the solid action transports the agent to the state to the right, while taking the dashed action resets the agent to the start state 1 and it has to re-start from 1 trying to get

¹ It is not a real restriction to assume $r_t \in [0, 1]$ since any bounded reward function can be shifted and rescaled to the range $[0, 1]$ without affecting the optimal policy (Ng et al, 1999).

↙ =
much
Lesser
than

to the goal state. To simplify exposition, we use $\gamma = 1$, and $R(s,a) = -1$ for all state–actions unless $s = G$, in which case $R(s,a) = 0$.

An agent that uses ε -greedy exploration rule will, at each timestep, be reset to the start state with probability at least $\varepsilon/2$. Therefore, the probability that the agent always chooses the solid action until it reaches the goal is $(1 - \varepsilon/2)^N$, which implies the value of state N is exponentially small in N , given any fixed value of ε . In contrast, the optimal policy requires only N steps, and the provably efficient algorithms described later in this chapter requires only a small number of steps (polynomial in N) to find the goal as well as the optimal policy. \square

This example demonstrates that a poor exploration rule may be exponentially inefficient, as can be shown for many other heuristics. Furthermore, inefficient exploration can have a substantial impact in practice, as illustrated by a number of empirical studies where techniques covered in this chapter can lead to smarter exploration schemes that outperform popular heuristics (see, e.g., Brunskill et al (2009), Li and Littman (2010), and Nouri and Littman (2009, 2010)). Such problems raise the critical need for designing algorithms that are *provably* efficient, which is the focus of the present chapter.

6.3 Formalizing Exploration Efficiency

Before developing provably efficient algorithms, one must define what it means by efficient exploration. A few choices are surveyed below.

6.3.1 Sample Complexity of Exploration and PAC-MDP

In measuring the *sample complexity* of exploration (or *sample complexity* for short) of a reinforcement-learning algorithm \mathbf{A} , it is natural to relate the complexity to the number of steps in which the algorithm does *not* choose near-optimal actions. To do so, we treat \mathbf{A} as a non-stationary policy that maps histories to actions. Two additional inputs are needed: $\varepsilon > 0$ and $\delta \in (0,1)$. The *precision parameter*, ε , controls the quality of behavior we require of the algorithm (*i.e.*, how close to optimality do we desire to be). The *confidence parameter*, δ , measures how certain we want to be of the algorithm’s performance. As both parameters decrease, greater exploration and learning is necessary as more is expected of the algorithms.

Definition 6.2. (Kakade, 2003) Let $c_t = (s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_t)$ be a path generated by executing a reinforcement-learning algorithm \mathbf{A} in an MDP M up to timestep t . The algorithm is viewed as a non-stationary policy, denoted \mathbf{A}_t at timestep t . Let (r_t, r_{t+1}, \dots) be the random sequence of reward received by \mathbf{A} after timestep t . The state value at s_t , denoted $V^{\mathbf{A}_t}(s_t)$, is the expectation of the discounted cumulative reward, $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$. Given a fixed $\varepsilon > 0$, the *sample complexity* of \mathbf{A} is

the number of timesteps τ such that the policy at timestep τ , A_τ , is not ε -optimal: $V^{A_\tau}(s_\tau) \leq V^*(s_\tau) - \varepsilon$.

The sample complexity of an algorithm A is defined using the infinite-length online interaction between an agent running A and an environment. If we view it a “mistake” when $V^{A_t}(s_t) \leq V^*(s_t) - \varepsilon$ happens, then sample complexity measures the total number of mistakes an RL algorithm makes during its whole run. We place no limitations on when the algorithm makes a mistake during an infinite-length interaction with the MDP. This freedom is essential for RL because, in a stochastic MDP, the agent does not have absolute control of the sequence of states it will visit, and so a mistake may happen at any time during the interaction, depending on when the agent is stochastically transitioned to those “bad” states. Intuitively, we prefer algorithms with smaller sample complexity, which motivates the following notation of exploration efficiency:

Definition 6.3. (Strehl et al, 2006a,b) An algorithm A is said to be *PAC-MDP (Probably Approximately Correct in Markov Decision Processes)* in an MDP M if, for any $\varepsilon > 0$ and $0 < \delta < 1$, the sample complexity of A is bounded by some function polynomial in the relevant quantities ($1/\varepsilon$, $1/\delta$, $1/(1-\gamma)$, and $|M|$) with probability at least $1 - \delta$, where $|M|$ measures the complexity of the MDP M . Usually, $|M|$ is polynomially related to the number of parameters describing M ; for instance, it is the numbers of states and actions for finite MDPs, and it is the number of parameters if the MDP can be described in a parametric way. Furthermore, the algorithm is called *efficiently PAC-MDP* if its computational and sample complexities are both polynomial.

A discussion of the parameters involved in Definition 6.3 is in order:

- The precision parameter, ε , specifies the level of optimality we want the algorithm to achieve, and a mistake is incurred when the non-stationary policy, A_t , is not ε -optimal. As in supervised learning, we cannot guarantee $\varepsilon = 0$ because any finite set of samples cannot accurately reveal the complete dynamics of an MDP when it is stochastic.
- The confidence parameter, δ , measures how certain we want to be of the algorithm’s sample complexity of exploration. As in supervised learning, we cannot guarantee $\delta = 0$ because there is always a non-zero, albeit tiny, probability that the observed interactions/samples are not representative of the true dynamics of the MDP and thus misleads the agent to act sub-optimally.
- The dependence on $1/(1-\gamma)$ is necessary and common in reinforcement learning. In many RL analyses, $1/(1-\gamma)$ plays a role that is roughly the effective decision horizon for value functions and policies. The larger this horizon is, the more we expect the sample complexity to be.
- The quantity $|M|$ of an MDP M is used to measure how large the MDP is. It can often be defined naturally, and should be made clear when we analyze the sample complexity for specific classes of MDPs.

The above definition of sample complexity was first used in the analysis of Rmax (Kakade, 2003), and will be the focus of the present chapter. It is worth mentioning that the analyses of E³ family of algorithms (Kakade et al, 2003; Kearns and Koller, 1999; Kearns and Singh, 2002) use slightly different definitions of efficient learning. In particular, these algorithms are required to halt after a polynomial amount of time and output a near-optimal policy for the *last* state, with high probability. Our analyses here are essentially equivalent, but simpler in the sense that mixing-time arguments can be avoided.

The notion of PAC-MDP is quite general, avoiding the need for assumptions such as reset (Fiechter, 1994, 1997), parallel sampling oracle (Even-Dar et al, 2002; Kearns and Singh, 1999), or reachability of the transition matrices (see the next subsection).² On the other hand, as will be shown later, the criterion of achieving small sample complexity of exploration is tractable in many challenging problems.

6.3.2 Regret Minimization

Another important notion of exploration efficiency is *regret*, which is more naturally defined in the average-reward criterion. Usually, the T -step regret, denoted $L^A(T)$, of an algorithm A is the difference between the T -step cumulative reward obtained by an optimal policy and that by an RL algorithm. A formal definition follows.

Definition 6.4. Given an MDP M and a start state s_1 , the T -step return of an algorithm A is the total reward the agent collects by starting in s_1 and following A :

$$R^A(M, s_1, T) \stackrel{\text{def}}{=} \sum_{t=1}^T r_t.$$

The average per-step reward of A is then given by

$$\rho^A(M, s_1) \stackrel{\text{def}}{=} \frac{1}{T} \mathbf{E} [R^A(M, s_1, T)].$$

Similar quantities may be defined for static policies π : $R^\pi(M, s_1, T)$ and $\rho^\pi(M, s_1)$.

Under certain assumptions on the transition structure of M such as communicating (Puterman, 1994), $\rho^\pi(M, s_1)$ is maximized by a static policy π^* and the maximum is independent of the start state s_1 ; define $\rho^* \stackrel{\text{def}}{=} \max_\pi \rho^\pi(M, s_1)$. The T -step regret of an algorithm A is naturally defined as follows

$$L^A(T) \stackrel{\text{def}}{=} T\rho^* - R^A(M, s_1, T). \quad (6.1)$$

² It should be noted that current PAC-MDP analysis relies heavily on discounting, which essentially makes the problem a finite-horizon one. This issue will be made explicit in the proof of Theorem 6.1.

P^* =
 max.
 average
 $P = R$ step
 reward

Naturally, regret is exactly what a reward-maximizing algorithm tries to minimize. If $L^A(T) = o(T)$, it can be concluded that the non-stationary policy followed by the agent, A_t , becomes indistinguishable, in terms of cumulative reward, from the optimal policy in the long run. The smaller $L^A(T)$ is, the more efficient the algorithm explores and learns. While regret minimization is a more direct objective for online reinforcement learning, algorithms with sub-linear regret may not be PAC-MDP in general.³ On the other hand, a PAC-MDP algorithm may suffer non-diminishing regret because the algorithm tries to optimize its rewards on the states it visits rather than on states visited by an optimal policy (Jaksch et al, 2010, Footnote 4).

However, non-trivial regret bounds have to rely on certain *reachability* assumptions on the transition structure of the underlying MDP:

Example 6.2. Consider the following MDP with two actions: if the agent takes the right action at $t = 1$, it reaches a “heaven” region S_1 of good states where rewards are always 1; otherwise, it reaches a “hell” region S_0 of states with zero reward everywhere. If it is impossible to navigate from states in S_0 to S_1 , a suboptimal action made by the agent at $t = 1$ can never be made up for, and so the regret grows linearly: $L^A(T) = \Theta(T)$. □

For this reason, all regret analyses in the literature rely on various sorts of reachability assumptions. In particular, assuming irreducibility of the transition matrices, an asymptotically logarithmic regret is possible (Burnetas and Katehakis, 1997; Tewari and Bartlett, 2008): $L^A(T) \leq C \cdot \ln T$ for some constant C . Unfortunately, the value of C depends on the dynamics of the underlying MDP and in fact can be arbitrarily large.

A weaker assumption is introduced by Jaksch et al (2010), whose **UCRL2** algorithm has a regret of $\tilde{O}\left(D|\mathbf{S}|\sqrt{|\mathbf{A}|T}\right)$, where the *diameter parameter* D is defined such that every state can be reached by any other state in at most D steps on average. More recently, regret analysis is applied to weakly communicating MDPs for the **REGAL** algorithm of Bartlett and Tewari (2009), yielding a regret bound of $\tilde{O}\left(H|\mathbf{S}|\sqrt{|\mathbf{A}|T}\right)$, where H is the *span* of the optimal value function. The bound improves that of **UCRL2** since $H \leq D$ for all weakly communicating MDPs.

Interestingly, these no-regret algorithms also implement the same principle for exploration (known as “optimism in the face of uncertainty”) as many PAC-MDP algorithms (Sections 6.4–6.6). For instance, both **UCRL2** and **MBIE** (Section 6.5) use interval estimation to maintain a set \mathbf{M} of MDPs that contain the underlying MDP with high probability, and then follow the optimal policy of the most optimistic model in \mathbf{M} . The main difference is in their analysis of how fast \mathbf{M} shrinks and how it relates to regret and sample complexity, respectively.

Finally, we would like to emphasize the flexibility of regret as a theoretical concept that can be used beyond traditional MDPs. For instance, motivated by competitive analysis in adversarial online learning with bandit feedback, recent efforts

³ An example is an exploration scheme for a one-state MDP (also known as a multi-armed bandit) that is greedy in the limit while ensuring infinite exploration at the same time (Robbins, 1952, Section 2): the regret is sub-linear in T but the sample complexity is ∞ .

were made to derive strong regret bounds even when the reward function of the MDP may change in an adaptively adversarial manner in every step, given complete knowledge of the fixed transition probabilities (see, *e.g.*, Neu et al (2011) and the references therein).

6.3.3 Average Loss

In the heaven-or-hell example in Section 6.3.2, the sample complexity of any algorithm in this problem is trivially 1, in contrast to the poor linear regret bound. The drastic difference highlights a fundamental difference between sample complexity and regret: the former is defined on the states visited by the algorithm, while the latter is defined with respect to the rewards along the trajectory of the optimal policy. One may argue that this MDP is difficult to solve since mistakes in taking actions cannot be recovered later, so the poor guarantee in terms of regret seems more consistent with the hardness of the problem than, say, sample complexity. However, mistakes are unavoidable for a trial-and-error agent as in reinforcement learning. Thus it seems more natural to evaluate an algorithm *conditioned on the history*. In this sense, the heaven-or-hell MDP is easy to solve since there is only one place where the agent can make a mistake.

One may define the loss in cumulative rewards in a different way to avoid reachability assumptions needed by regret analysis. One possibility is to use the assumption of reset (or episodic tasks), with which the agent can go back to the same start state periodically (Fiechter, 1994, 1997). However, resets are usually not available in most online reinforcement-learning problems.

Another possibility, called the average loss (Strehl and Littman, 2008a), compares the loss in cumulative reward of an agent on the sequence of states the agent actually visits:

Definition 6.5. Fix $T \in \mathbb{N}$ and run an RL algorithm \mathbf{A} for T steps. The *instantaneous loss* l_t at timestep t is the difference between the optimal state value and the cumulative discounted return:

$$l_t \stackrel{\text{def}}{=} V^*(s_t) - \sum_{\tau=t}^T \gamma^{\tau-t} r_\tau.$$

The *average loss* is then defined by

$$g(T) \stackrel{\text{def}}{=} \frac{1}{T} \sum_{t=1}^T l_t.$$

The algorithm \mathbf{A} is probably approximately correct in the average loss criterion if, for any $\varepsilon > 0, \delta \in (0, 1)$, we can choose T , polynomial in the relevant quantities ($|\mathbf{S}|$, $|\mathbf{A}|$, $1/(1-\gamma)$, $1/\varepsilon$, and $1/\delta$), such that $g(T) \leq \varepsilon$ with probability at least $1 - \delta$.

As in the definition of sample complexity, reachability assumptions are avoided as the quantity $g(T)$ is defined on the sequence of states actually visited by the algorithm **A**. Learnability in the average loss model is weaker since it does not imply diminishing regret (Jaksch et al, 2010, Footnote 4). Furthermore, it can be shown that every PAC-MDP algorithm is probably approximately correct in the average loss criterion (Strehl and Littman, 2008a).

6.3.4 Bayesian Framework

In a Bayesian setting, an agent may assume a prior distribution over parameters that specify an MDP (including the transition probabilities and reward function), and maintain the posterior distribution using Bayes' rule during online interaction with the environment. Then, we could ask for the *Bayes optimal* policy that maximizes expected cumulative reward with respect to the posterior distribution. In a sense, there is no need for explicit exploration in this setting, since uncertainty in the MDP model is completely captured by the posterior distribution by definition. Consequently, the agent can simply follow the Bayes optimal policy (Duff, 2002).

A major limitation of Bayesian approaches is their high computation complexity. In general, computing the Bayes optimal policy is often intractable, making approximation inevitable (see, e.g., Duff (2002)). It remains an active research area to develop efficient algorithms to approximate the Bayes optimal policy (Poupart et al, 2006; Kolter and Ng, 2009).

While computationally expensive, Bayesian RL possesses the important benefit of being able to use a prior distribution of models to guide exploration. With an appropriate prior, Bayesian exploration can have superior performance. In contrast, algorithms with small sample complexity or low regret are often developed to handle *worst-case* scenarios and are often conservative in practice when used naively. However, the benefits of Bayesian approaches come with a price: they generally lack strong theoretical guarantees, especially when the prior distribution is *mis-specified*. For example, Bayesian algorithms are not PAC-MDP in general, or may even converge to a sub-optimal policy in the limit; examples are found in Kolter and Ng (2009, Theorem 2) and Li (2009, Example 9).

Finally, we note that it is possible to combine some of the techniques from Bayesian and PAC-MDP methods to obtain the best of both worlds. For instance, the analytic tools developed for PAC-MDP algorithms are used to derive the **BEB** algorithm that approximates the Bayes optimal policy except for polynomially many steps (Kolter and Ng, 2009). As another example, the notion of known state–actions is combined with the posterior distribution of models to yield a randomized PAC-MDP algorithm (**BOSS**) that is able to use prior knowledge about MDP models (Asmuth et al, 2009).

6.4 A Generic PAC-MDP Theorem

This section gives a generic theorem for proving polynomial sample complexity for a class of online RL algorithms. This result serves as a basic tool for further investigation of model-based and model-free approaches in this chapter.

Consider an RL algorithms \mathbf{A} that maintains an estimated state-action value function $Q(\cdot, \cdot)$, and let Q_t denote the value of Q immediately before the t -th action of the agent. We say that \mathbf{A} is *greedy* if it always chooses an action that maximizes its current value function estimate; namely, $a_t = \arg \max_{a \in \mathbf{A}} Q_t(s_t, a)$, where s_t is the t -th state reached by the agent. For convenience, define $V_t(s) \stackrel{\text{def}}{=} \max_{a \in \mathbf{A}} Q_t(s, a)$. For our discussions, two important definitions are needed, in which we denote by $\mathbf{K} \subseteq \mathbf{S} \times \mathbf{A}$ an arbitrary set of state-actions. While \mathbf{K} may be arbitrarily defined, it is often understood to be a subset that the agent already “knows” and need not be explored. Finally, a state s is called “known” if $(s, a) \in \mathbf{K}$ for all $a \in \mathbf{A}$.

Definition 6.6. We define E_t to be the event, called the *escape event (from \mathbf{K})*, that some state-action $(s, a) \notin \mathbf{K}$ is experienced by algorithm \mathbf{A} at timestep t .

Definition 6.7. Given an MDP $M = \langle \mathbf{S}, \mathbf{A}, T, R, \gamma \rangle$ and a state-action value function Q , the *known state-action MDP with respect to \mathbf{K}* , denoted $M_{\mathbf{K}}$, is an MDP with the same state and action sets but different reward and transition functions:

$$T_{\mathbf{K}}(s, a, s') = \begin{cases} T(s, a, s') & \text{if } (s, a) \in \mathbf{K} \\ \mathbb{I}(s' = s) & \text{otherwise} \end{cases} \quad (6.2)$$

$$R_{\mathbf{K}}(s, a) = \begin{cases} R(s, a) & \text{if } (s, a) \in \mathbf{K} \\ (1 - \gamma)Q(s, a) & \text{otherwise,} \end{cases} \quad (6.3)$$

where $\mathbb{I}(s' = s)$ is the indicator function. If we replace the true dynamics T and R by respective estimates \hat{T} and \hat{R} in the right-hand side of Equations 6.2 and 6.3, then the resulting MDP, denoted $\hat{M}_{\mathbf{K}}$, is called an *empirical known state-action MDP with respect to \mathbf{K}* .

Intuitively, the known state-action MDP $M_{\mathbf{K}}$ is an optimistic model of the true MDP M as long as Q is optimistic (namely, $Q(s, a) \geq Q^*(s, a)$ for all (s, a)). Furthermore, the two MDPs’ dynamics agree on state-actions in \mathbf{K} . Consider two cases:

1. If it takes too many steps for the agent to navigate from its current state s to some unknown state by any policy, the value of exploration is small due to γ -discounting. In other words, all unknown states are essentially irrelevant to optimal action selection in state s , so the agent may just follow the optimal policy of $M_{\mathbf{K}}$, which is guaranteed to be near-optimal for s in M .
2. If, on the other hand, an unknown state is close to the current state, the construction of $M_{\mathbf{K}}$ (assuming Q is optimistic) will encourage the agent to navigate to the unknown states (the “escape” event) for exploration. The policy in the current state may not be near-optimal.

Hence, the number of times the second case happens is linked directly to the sample complexity of the algorithm. The known state-action MDP is the key concept to balance exploration and exploitation in an elegant way.

The intuition above is formalized in the following generic theorem (Li, 2009), which slightly improves the original result (Strehl et al, 2006a). It provides a common basis for all our sample-complexity analyses later in this chapter.

Theorem 6.1. *Let $\mathbf{A}(\varepsilon, \delta)$ be an algorithm that takes ε and δ as inputs (in addition to other algorithm-specific inputs), acts greedily according to its estimated state-action value function, denoted Q_t at timestep t . Suppose that on every timestep t , there exists a set \mathbf{K}_t of state-actions that depends only on the agent's history up to timestep t . We assume that $\mathbf{K}_t = \mathbf{K}_{t+1}$ unless, during timestep t , an update to some state-action value occurs or the escape event E_t happens. Let $M_{\mathbf{K}_t}$ be the known state-action MDP (defined using \mathbf{K}_t and Q_t in Definition 6.7) and π_t be the greedy policy with respect to Q_t . Suppose that for any inputs ε and δ , with probability at least $1 - \delta/2$, the following conditions hold for all timesteps t :*

1. (Optimism) $V_t(s_t) \geq V^*(s_t) - \varepsilon/4$,
2. (Accuracy) $V_t(s_t) - V_{M_{\mathbf{K}_t}}^{\pi_t}(s_t) \leq \varepsilon/4$, and
3. (Bounded Surprises) The total number of updates of action-value estimates plus the number of times the escape event from \mathbf{K}_t , E_t , can occur is bounded by a function $\zeta(\varepsilon, \delta, |M|)$.

Then, with probability at least $1 - \delta$, the sample complexity of $\mathbf{A}(\varepsilon, \delta)$ is

$$O\left(\frac{V_{\max}}{\varepsilon(1-\gamma)} \left(\zeta(\varepsilon, \delta, |M|) + \ln \frac{1}{\delta}\right) \ln \frac{1}{\varepsilon(1-\gamma)}\right).$$

Proof. (sketch) The proof consists of two major parts. First, let W denote the event that, after following the non-stationary policy \mathbf{A}_t from state s_t in M for $H = \frac{1}{1-\gamma} \ln \frac{4}{\varepsilon(1-\gamma)}$ timesteps, one of the two following events occur: (a) the state-action value function estimate is changed for some (s, a) ; or (b) an escape event from \mathbf{K}_t happens; namely, some $(s, a) \notin \mathbf{K}_t$ is experienced. The optimism and accuracy conditions then imply an enlightening result in the analysis, known as the “Implicit Explore or Exploit” lemma (Kakade, 2003; Kearns and Singh, 2002):

$$V^{\mathbf{A}_t}(s_t) \geq V^*(s_t) - 3\varepsilon/4 - \Pr(W)V_{\max}.$$

In other words, if $\Pr(W) < \frac{\varepsilon}{4V_{\max}}$, \mathbf{A}_t is ε -optimal in s_t (exploitation); otherwise, the agent will experience event W with probability at least $\frac{\varepsilon}{4V_{\max}}$ (exploration).

The second part of the proof bounds the number of times when $\Pr(W) \geq \frac{\varepsilon}{4V_{\max}}$ happens. To do so, we split the entire trajectory into pieces of length H , each of which starts in state s_{iH+1} for $i \in \mathbb{N}$: $(s_1, \dots, s_H), (s_{H+1}, \dots, s_{2H}), \dots$. We then view as Bernoulli trials the H -step sub-trajectories of the agent starting in s_{iH+1} and following algorithm \mathbf{A} . The trial succeeds if event W happens. Clearly, outcomes of these Bernoulli trials are independent of each other, conditioned on the starting states of the sub-trajectories. Since each of the Bernoulli trials succeeds with probability at least $\frac{\varepsilon}{4V_{\max}}$ and the total number of successes is at most $\zeta(\varepsilon, \delta, |M|)$ (due to the

bounded-surprises condition), it can be shown, using Azuma's inequality, that, with probability at least $1 - \delta/2$, there are at most $\frac{8HV_{\max}}{\varepsilon} (\zeta(\varepsilon, \delta) + \ln \frac{2}{\delta})$ steps in which $\Pr(W) \geq \frac{\varepsilon}{4V_{\max}}$.

Finally, the theorem follows from an application of a union bound to the two cases, each of which can happen with probability at most $\delta/2$: (i) the three conditions may fail to hold, and (ii) the second part of the proof above may also fail. \square

Theorem 6.1 is a powerful theoretical tool. It provides a convenient way to prove an algorithm is PAC-MDP, as shown in the following two sections for both model-based and model-free ones.

6.5 Model-Based Approaches

This section presents model-based PAC-MDP algorithms. These algorithms usually estimate parameters of an unknown MDP explicitly and then compute the optimal policy of the estimated MDP model. The simulation lemma (see, e.g., (Kearns and Singh, 2002)) states that MDPs with similar dynamics have similar value functions, and hence ensures that the policy will be near-optimal in the true MDP as long as the estimated MDP model is learned to sufficient accuracy.

Model-based algorithms usually require solving an MDP as a sub-routine and hence are computationally expensive. However, by maintaining more information about the problem, model-based algorithms are often found more effective in practice (see, e.g., Moore and Atkeson (1993)). Indeed, the first PAC-MDP algorithms, including **E**³ (Kearns and Singh, 2002) and **Rmax** (Brafman and Tennenholz, 2002), explicitly distinguish the set of “known states,” in which accurate reward/transition predictions can be made, from the set of “unknown states.” As it turns out, this technique can be extended to general MDPs based on a novel learning model (Section 6.5.2).

6.5.1 Rmax

Algorithm 12 gives complete pseudocode for **Rmax**. Two critical ideas are used in the design of **Rmax**. One is the distinction between known states—states where the transition distribution and rewards can be accurately inferred from observed transitions—and unknown states. The other is the principle for exploration known as “optimism in the face of uncertainty” (Brafman and Tennenholz, 2002).

The first key component in **Rmax** is the notion of known state–actions. Suppose an action $a \in \mathbf{A}$ has been taken m times in a state $s \in \mathbf{S}$. Let $r[i]$ and $s'[i]$ denote the i -th observed reward and next state, respectively, for $i = 1, 2, \dots, m$. A maximum-likelihood estimate of the reward and transition function for (s, a) is given by:

$$\hat{T}(s, a, s') = |\{i \mid s'[i] = s'\}| / m, \quad \forall s' \in \mathbf{S} \quad (6.4)$$

$$\hat{R}(s, a) = (r[1] + r[2] + \dots + r[m]) / m. \quad (6.5)$$

```

0: Inputs:  $\mathbf{S}, \mathbf{A}, \gamma, \varepsilon, \delta, m$ .
1: Initialize counter  $c(s,a) \leftarrow 0$  for all  $(s,a) \in \mathbf{S} \times \mathbf{A}$ .
2: Initialize the empirical known state-action MDP  $\hat{M} = \langle \mathbf{S}, \mathbf{A}, \hat{T}, \hat{R}, \gamma \rangle$ :

$$\hat{T}(s,a,s') = \mathbb{I}(s' = s), \quad \hat{R}(s,a) = V_{\max}(1 - \gamma).$$

3: for all timesteps  $t = 1, 2, 3, \dots$  do
4:   Compute the optimal state-action value function of  $\hat{M}$ , denoted  $Q_t$ .
5:   Observe the current state  $s_t$ , take a greedy action  $a_t = \arg \max_{a \in \mathbf{A}} Q_t(s_t, a)$ ,
   receive reward  $r_t$ , and transition to the next state  $s_{t+1}$ .
6:    $c(s_t, a_t) \leftarrow c(s_t, a_t) + 1$ 
7:   if  $c(s_t, a_t) = m$  then
8:     Redefine  $\hat{T}(s_t, a_t, \cdot)$  and  $\hat{R}(s_t, a_t)$  using Equations 6.4 and 6.5.

```

Algorithm 12. Rmax

Intuitively, we expect these estimates to converge almost surely to their true values, $R(s,a)$ and $T(s,a,s')$, respectively as m becomes large. When m is large enough (in a sense made precise soon), the state-action (s,a) is called “known” since we have an accurate estimate of the reward and transition probabilities for it; otherwise, (s,a) is “unknown.” Let \mathbf{K}_t be the set of known state-actions at timestep t (before the t -th action is taken). Clearly, $\mathbf{K}_1 = \emptyset$ and $\mathbf{K}_t \subseteq \mathbf{K}_{t+1}$ for all t .

The second key component in Rmax explicitly deals with exploration. According to the simulation lemma, once all state-actions become known, the agent is able to approximate the unknown MDP with high accuracy and thus act near-optimally. It is thus natural to encourage visitation to unknown state-actions. This goal is achieved by assigning the optimistic value V_{\max} to unknown state-actions such that, for all timestep t , $Q_t(s,a) = V_{\max}$ if $(s,a) \notin \mathbf{K}_t$. During execution of the algorithm, if action a_t has been tried m times in state s_t , then $\mathbf{K}_{t+1} = \mathbf{K}_t \cup \{(s_t, a_t)\}$. It is easy to see that the empirical state-action MDP solved by Rmax at timestep t coincides with the empirical known state-action MDP in Definition 6.7 (with \mathbf{K}_t and Q_t).

Theorem 6.1 gives a convenient way to show Rmax is PAC-MDP. While a complete proof is available in the literature (Kakade, 2003; Strehl et al, 2009), we give a proof sketch that highlights some of the important steps in the analysis.

Theorem 6.2. Rmax is PAC-MDP with the following sample complexity:

$$\tilde{O}\left(\frac{|\mathbf{S}|^2 |\mathbf{A}| V_{\max}^3}{\varepsilon^3 (1 - \gamma)^3}\right).$$

Proof. (sketch) It suffices to verify the three conditions in Theorem 6.1 are satisfied by Rmax. Using Chernoff/Hoeffding’s inequality, we can show that the maximum-likelihood estimates in Equations 6.4 and 6.5 are very accurate for large m : more precisely, with probability at least $1 - \delta$, we have

$$\begin{aligned} |\hat{R}(s,a) - R(s,a)| &= O(\varepsilon(1-\gamma)) \\ \sum_{s' \in S} |\hat{T}(s,a,s') - T(s,a,s')| &= O\left(\frac{\varepsilon(1-\gamma)}{V_{\max}}\right) \end{aligned}$$

for any (s,a) as long as $m \geq m_0$ for some $m_0 = \tilde{O}\left(\frac{|S|V_{\max}^2}{\varepsilon^2(1-\gamma)^2}\right)$. With these highly accurate estimates of reward and transition probabilities, the optimism and accuracy conditions in Theorem 6.1 then follow immediately by the simulation lemma. Furthermore, since any unknown state-action can be experienced at most m times (after which it will become known), the bounded-surprises condition also holds with

$$\zeta(\varepsilon, \delta, |M|) = |S||A|m = \tilde{O}\left(\frac{|S|^2|A|V_{\max}^2}{\varepsilon^2(1-\gamma)^2}\right).$$

The sample complexity of **Rmax** then follows immediately. \square

The **Rmax** algorithm and its sample complexity analysis may be improved in various ways. For instance, the *binary* concept of knownness of a state-action may be replaced by the use of interval estimation that smoothly quantifies the prediction uncertainty in the maximum-likelihood estimates, yielding the **MBIE** algorithm (Strehl and Littman, 2008a). Furthermore, it is possible to replace the constant optimistic value V_{\max} by a non-constant, optimistic value function to gain further improvement in the sample complexity bound (Strehl et al, 2009; Szita and Lőrincz, 2008). More importantly, we will show in the next subsection how to extend **Rmax** from finite MDPs to general, potentially infinite, MDPs with help of a novel supervised-learning model.

Finally, we note that a variant of **Rmax**, known as **MoRmax**, is recently proposed (Szita and Szepesvári, 2010) with a sample complexity of $\tilde{O}\left(\frac{|S||A|V_{\max}^2}{\varepsilon^2(1-\gamma)^4}\right)$. Different from **Rmax** which freezes its transition/reward estimates for a state-action once it becomes known, **MoRmax** sometimes re-estimates these quantities when new m -tuples of samples are collected. While this algorithm and its analysis are more complicated, its sample complexity significantly improves over that of **Rmax** in terms of $|S|$, $1/\varepsilon$, and $1/V_{\max}$, at a cost of an additional factor of $1/(1-\gamma)$. Its dependence on $|S|$ matches the currently best lower bound (Li, 2009).

6.5.2 A Generalization of Rmax

Rmax is not applicable to continuous-state MDPs that are common in some applications. Furthermore, its sample complexity has an explicit dependence on the number of state-actions in the MDP, which implies that it is inefficient even in finite MDPs when the state or action space is large. As in supervised learning, *generalization* is needed to avoid explicit enumeration of every state-action, but generalization can also make exploration more challenging. Below, we first introduce a novel model for supervised learning, which allows one to extend **Rmax** to arbitrary MDPs.

6.5.2.1 Knows What It Knows

As explained before, a key property of **Rmax** is to distinguish known state–actions from unknown ones according to the uncertainty in its prediction about the dynamics in that state–action. This knowledge provides a useful mechanism for efficient exploration. Motivated by this observation, the “*Knows What It Knows*” (or **KWIK**) framework is proposed to capture the essence of quantifying estimation uncertainty (Li et al, 2011).

A KWIK problem is specified by a five-tuple: $P = \langle \mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{H}, h^* \rangle$, where: \mathbf{X} is an *input set*; \mathbf{Y} is an *output set*; \mathbf{Z} is an *observation set*; $\mathbf{H} \subseteq \mathbf{Y}^\mathbf{X}$ is a class of hypotheses mapping \mathbf{X} to \mathbf{Y} ; and $h^* \in \mathbf{Y}^\mathbf{X}$ is an unknown *target function*. We call a KWIK problem “realizable” if $h^* \in \mathbf{H}$; otherwise, it is called “agnostic.”

Two parties are involved in a KWIK learning process. The *learner* runs a learning algorithm and makes predictions; the *environment*, which represents an instance of a KWIK learning problem, provides the learner with inputs and observations. These two parties interact according to the following protocol:

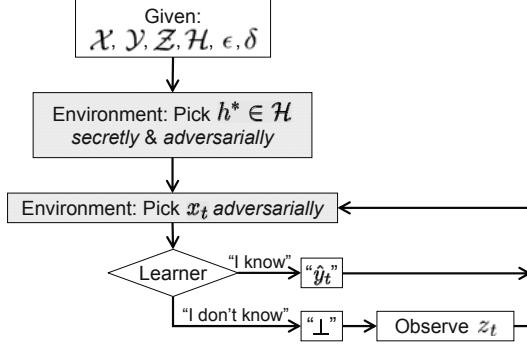
Definition 6.8. A KWIK learning process proceeds as follows (Figure 6.2):

1. The input set \mathbf{X} , output set \mathbf{Y} , observation set \mathbf{Z} , hypothesis class \mathbf{H} , accuracy parameter $\varepsilon > 0$, and confidence parameter $\delta \in (0,1)$ are known to both the learner and the environment.
2. The environment selects a target function $h^* \in \mathbf{Y}^\mathbf{X}$ secretly and adversarially.
3. At timestep $t = 1, 2, 3, \dots$
 - The environment selects an input $x_t \in \mathbf{X}$ in an *arbitrary* way and informs the learner. The target value $y_t = h^*(x_t)$ is unknown to the learner. Note that the choice of x_t may depend on the history of the interaction.
 - The learner predicts an output $\hat{y}_t \in \mathbf{Y} \cup \{\perp\}$, where \perp is a special symbol that is not in \mathbf{Y} . We call \hat{y}_t *valid* if $\hat{y}_t \neq \perp$.
 - If $\hat{y}_t = \perp$, the learner makes an observation $z_t \in \mathbf{Z}$ of the output. In the deterministic case, $z_t = y_t$. In the stochastic case, the relation between z_t and y_t is problem specific. For example, in the Bernoulli case where $\mathbf{Y} = [0,1]$ and $\mathbf{Z} = \{0,1\}$, we have $z_t = 1$ with probability y_t and 0 otherwise.

Definition 6.9. Let $\mathbf{H} \subseteq \mathbf{Y}^\mathbf{X}$ be a hypothesis class. We say that \mathbf{H} is *KWIK-learnable* if there exists an algorithm \mathbf{A} with the following property: for any $\varepsilon > 0$ and $\delta \in (0,1)$, the following two requirements are satisfied with probability at least $1 - \delta$ in a whole run of \mathbf{A} according to the KWIK protocol above:

1. (*Accuracy Requirement*) If $h^* \in \mathbf{H}$, then all non- \perp predictions must be ε -accurate; that is, $|\hat{y}_t - y_t| < \varepsilon$ whenever $\hat{y}_t \neq \perp$.⁴

⁴ Here, $|\hat{y} - y|$ may be any nonnegative-valued function that measures the *discrepancy* or *distance* between \hat{y} and y . To make notation more intuitive, we use the operator “ $-$ ” in a broad sense and it is not restricted to algebraic subtraction. For instance: (i) in the special situation of $\mathbf{Y} \subseteq \mathbb{R}$, $|\cdot|$ is understood as the absolute value; (ii) when $\mathbf{Y} \subseteq \mathbb{R}^k$, $|\cdot|$ may mean vector norm; and (iii) when y and \hat{y} are probability distributions, $|\cdot|$ may be used for total variation.

**Fig. 6.2** KWIK protocol

2. (*Sample Complexity Requirement*) The total number of \perp s predicted during the whole run, denoted $B(\varepsilon, \delta, \dim(\mathbf{H}))$, is bounded by a function polynomial in $1/\varepsilon$, $1/\delta$, and $\dim(\mathbf{H})$, where $\dim(\mathbf{H})$ is a pre-defined nonnegative-valued function measuring the *dimension* or *complexity* of \mathbf{H} .

We call \mathbf{A} a *KWIK algorithm* and $B(\varepsilon, \delta, \dim(\mathbf{H}))$ a *KWIK bound* of \mathbf{A} . Furthermore, \mathbf{H} is *efficiently KWIK-learnable* if, in addition to the accuracy and sample-complexity requirements, the per-step time complexity of \mathbf{A} is polynomial in $1/\varepsilon$, $1/\delta$, and $\dim(\mathbf{H})$.

Although KWIK is a harder learning model than existing models like Probably Approximately Correct (Valiant, 1984) and Mistake Bound (Littlestone, 1987), a number of useful hypothesis classes are shown to be efficiently KWIK-learnable. Below, we review some representative examples from Li et al (2011).

Example 6.3. Consider the problem of learning a deterministic linear function $f(x) \stackrel{\text{def}}{=} x \cdot \theta^*$ for some unknown vector $\theta^* \in \mathbb{R}^d$: the input $x \in \mathbb{R}^d$ is a d -dimensional vector, and both the output and observation are $y = z = f(x)$. To KWIK-learn such linear functions, we may maintain a set \mathbf{D} of training examples, which is initialized to the empty set \emptyset prior to learning. On the t -th input x_t , let

$$\mathbf{D} = \{(v_1, f(v_1)), (v_2, f(v_2)), \dots, (v_k, f(v_k))\}$$

be the current set of training examples. The algorithm first detects if x_t is linearly independent of the previous inputs stored in \mathbf{D} . If x_t is linearly independent, then the algorithm predicts \perp , observes the output $y_t = f(x_t)$, and then expands the training set: $\mathbf{D} \leftarrow \mathbf{D} \cup \{(x_t, y_t)\}$. Otherwise, there exist k real numbers, a_1, a_2, \dots, a_k , such that $x_t = a_1 v_1 + a_2 v_2 + \dots + a_k v_k$. In the latter case, we can accurately predict the value of $f(x_t)$: $f(x_t) = a_1 f(v_1) + a_2 f(v_2) + \dots + a_k f(v_k)$. Furthermore, since \mathbf{D} can contain at most d linear independent inputs, the algorithm's KWIK bound is d . \square

Example 6.4. As an example of stochastic problems, consider the problem of learning a multinomial distribution over n elements, where each distribution p is specified by n non-negative numbers that sum up to unity: $p = (p_1, p_2, \dots, p_n)$ where $p_i \geq 0$ and $\sum_{i=1}^n p_i = 1$. Since there is a *single* multinomial distribution p^* to learn, the inputs are irrelevant, the output is always p^* , and the observation is one of the n elements randomly drawn according to p^* . *Total variation* may be used as the distance metric: $|\hat{p} - p| \stackrel{\text{def}}{=} \frac{1}{2} \sum_{i=1}^n |\hat{p}_i - p_i|$. For this problem, we can predict \perp and obtain random samples for the *first* m steps, and then always predict the maximum-likelihood estimate \hat{p} , which is accurate as long as m is at least $\frac{2n}{\varepsilon^2} \ln \frac{2n}{\delta}$ (Kakade, 2003, Lemma 8.5.5). This algorithm, known as **dice-learning** (named after the problem of estimating biases in rolling a die), has a KWIK bound of $\tilde{O}(n/\varepsilon^2)$. \square

Example 6.5. We now consider a stochastic version of Example 6.3. The setting is the same except that: (i) the observation is corrupted by white noise: $z_t = y_t + \eta_t$ with $\mathbf{E}[\eta_t] = 0$; and (ii) all quantities (including x_t , θ^* , and η_t) have bounded ℓ_2 norm. Although there is noise in observations, one may still quantify the prediction error of the least-squares estimation. Using a covering argument, the **noisy linear regression** algorithm (Li et al, 2011; Strehl and Littman, 2008b) is shown to have a KWIK bound of $\tilde{O}(d^3/\varepsilon^4)$. A similar result is obtained by a simpler algorithm based on ridge regression (Walsh et al, 2009). \square

Interestingly, in many cases, it is possible to decompose a complicated KWIK problem into simpler KWIK subproblems, and then design *meta algorithms* that combine the sub-algorithms to solve the original problem (Li et al, 2011). Examples of such decompositions are provided when we study a generalization of **Rmax** next.

6.5.2.2 KWIK-Rmax

KWIK provides a formal learning model for studying prediction algorithms with uncertainty awareness. It is natural to combine a KWIK algorithm for learning MDPs with the basic **Rmax** algorithm. The main result in this subsection is the following: if a class of MDPs can be KWIK-learned, then there exists an **Rmax**-style algorithm that is PAC-MDP for this class of MDPs.

We first define KWIK-learnability of a class of MDPs, which is motivated by the simulation lemma. For any set A , we denote by \mathbf{P}_A the set of probability distributions defined over A .

Definition 6.10. Fix the state space \mathbf{S} , action space \mathbf{A} , and discount factor γ .

1. Define $\mathbf{X} = \mathbf{S} \times \mathbf{A}$, $\mathbf{Y}_T = \mathbf{P}_{\mathbf{S}}$, and $\mathbf{Z}_T = \mathbf{S}$. Let $\mathbf{H}_T \subseteq \mathbf{Y}_T^{\mathbf{X}}$ be a set of transition functions of an MDP. \mathbf{H}_T is (*efficiently*) KWIK-learnable if in the accuracy requirement of Definition 6.9, $|\hat{T}(\cdot | s, a) - T(\cdot | s, a)|$ is interpreted as the ℓ_1 distance defined by:

$$\begin{cases} \sum_{s' \in \mathbf{S}} |\hat{T}(s' | s, a) - T(s' | s, a)| & \text{if } \mathbf{S} \text{ is countable} \\ \int_{s' \in \mathbf{S}} |\hat{T}(s' | s, a) - T(s' | s, a)| ds' & \text{otherwise.} \end{cases}$$

```

0: Inputs:  $\mathbf{S}, \mathbf{A}, \gamma, V_{\max}, \mathbf{A}_T$  (with parameters  $\varepsilon_T, \delta_T$ ),  $\mathbf{A}_R$  (with parameters  $\varepsilon_R, \delta_R$ ),  $\varepsilon_P$ 
1: for all timesteps  $t = 1, 2, 3, \dots$  do
2:   // Update the empirical known state-action MDP  $\hat{M} = \langle \mathbf{S}, \mathbf{A}, \hat{T}, \hat{R}, \gamma \rangle$ 
3:   for all  $(s, a) \in \mathbf{S} \times \mathbf{A}$  do
4:     if  $\mathbf{A}_T(s, a) = \perp$  or  $\mathbf{A}_R(s, a) = \perp$  then
5:        $\hat{T}(s'|s, a) = \begin{cases} 1 & \text{if } s' = s \\ 0 & \text{otherwise} \end{cases}$  and  $\hat{R}(s, a) = (1 - \gamma)V_{\max}$  .
6:     else
7:        $\hat{T}(\cdot|s, a) = \mathbf{A}_T(s, a)$  and  $\hat{R}(s, a) = \mathbf{A}_R(s, a)$ .
8:     Compute a near-optimal value function  $Q_t$  of  $\hat{M}$  such that  $|Q_t(s, a) - Q_{\hat{M}}^*(s, a)| \leq \varepsilon_P$ 
      for all  $(s, a)$ , where  $Q_{\hat{M}}^*$  is the optimal state-action value function of  $\hat{M}$ .
9:     Observe the current state  $s_t$ , take action  $a_t = \arg \max_{a \in \mathbf{A}} Q_t(s_t, a)$ , receive reward
       $r_t$ , and transition to the next state  $s_{t+1}$ .
10:    if  $\mathbf{A}_T(s_t, a_t) = \perp$  then
11:      Inform  $\mathbf{A}_T$  of the sample  $(s_t, a_t) \rightarrow s_{t+1}$ .
12:    if  $\mathbf{A}_R(s_t, a_t) = \perp$  then
13:      Inform  $\mathbf{A}_R$  of the sample  $s_t \rightarrow r_t$ .

```

Algorithm 13. KWIK-Rmax

2. Define $\mathbf{X} = \mathbf{S} \times \mathbf{A}$ and $\mathbf{Y}_R = [0, 1]$. Let $\mathbf{H}_R \subseteq \mathbf{Y}_R^\mathbf{X}$ be a set of reward functions of an MDP. \mathbf{H}_R is (*efficiently*) *KWIK-learnable* if in the accuracy requirement of Definition 6.9, $|\hat{R}(s, a) - R(s, a)|$ is interpreted as the absolute value.
3. Let $\mathbf{M} = \{\langle \mathbf{S}, \mathbf{A}, T, R, \gamma \rangle \mid T \in \mathbf{H}_T, R \in \mathbf{H}_R\}$ be a class of MDPs. \mathbf{M} is (*efficiently*) *KWIK-learnable* if both \mathbf{H}_T and \mathbf{H}_R are (*efficiently*) KWIK-learnable.

Put simply, we have defined a class of MDPs as KWIK-learnable if their transition and reward functions are KWIK-learnable. As shown below, KWIK-learnability of MDP classes suffices to guarantee PAC-MDP-ness of a **Rmax**-style algorithm.

A generic algorithm, which we call **KWIK-Rmax**, is given in Algorithm 13. Similar to **Rmax**, it explicitly distinguishes known state-actions from unknown ones. However, it does so with help from two KWIK algorithms, \mathbf{A}_T and \mathbf{A}_R , for learning the transition probabilities and reward function, respectively. The set of known state-actions are precisely those for which both KWIK algorithms make valid predictions. By definition, these predictions must be highly accurate and can be reliably used. As in **Rmax**, unknown state-actions are assigned an optimistic value V_{\max} to encourage exploration. Finally, the algorithm allows an approximation error when solving the empirical known state-action MDP. This is important since exact planning is expensive or even impossible when the MDP is large or continuous.

Algorithm 13 is presented to simplify exposition and highlight the fundamental algorithmic components. A few notes are in order when implementing the algorithm:

1. The definitions of \hat{T} and \hat{R} are *conceptual* rather than *operational*. For finite MDPs, one may represent \hat{T} by a matrix of size $O(|\mathbf{S}|^2 |A|)$ and \hat{R} by a vector of size $O(|\mathbf{S}| |\mathbf{A}|)$. For structured MDPs, more compact representations are possible. For instance, MDPs with linear dynamical systems may be represented by matrices of finite dimension (Strehl and Littman, 2008b; Brunskill et al, 2009), and factored-state MDPs can be represented by a dynamic Bayes net (Kearns and Koller, 1999).
2. It is unnecessary to update \hat{T} and \hat{R} and recompute Q_t for every timestep t . The known state-action MDP \hat{M} (and thus $Q_{\hat{M}}^*$ and Q_t) remains unchanged unless some unknown state-action becomes known. Therefore, one may update \hat{M} and Q_t only when A_T or A_R obtain new samples in lines 13 or 16.
3. It is unnecessary to compute Q_t for *all* (s, a) . In fact, according to Theorem 6.1, it suffices to guarantee that Q_t is ε_P -accurate in state s_t : $|Q_t(s_t, a) - Q_{\hat{M}}^*(s_t, a)| < \varepsilon_P$ for all $a \in \mathbf{A}$. This kind of *local* planning often requires much less computation than *global* planning.
4. Given the approximate MDP \hat{M} and the current state s_t , the algorithm computes a near-optimal action for s_t . This step can be done using dynamic programming for finite MDPs. In general, however, doing so is computationally expensive. Fortunately, recent advances in approximate local planning have made it possible for large-scale problems (Kearns et al, 2002; Kocsis and Szepesvári, 2006; Walsh et al, 2010a).

The sample complexity of **KWIK-Rmax** is given by the following theorem. It shows that the algorithm's sample complexity of exploration scales linearly with the KWIK bound of learning the MDP. For every KWIK-learnable class \mathbf{M} of MDPs, **KWIK-Rmax** is *automatically* instantiated to a PAC-MDP algorithm for \mathbf{M} with appropriate KWIK learners.

Theorem 6.3. *Let \mathbf{M} be a class of MDPs with state space \mathbf{S} and action space \mathbf{A} . If \mathbf{M} can be (efficiently) KWIK-learned by algorithms A_T (for transition functions) and A_R (for reward functions) with respective KWIK bounds B_T and B_R , then **KWIK-Rmax** is PAC-MDP in \mathbf{M} . In particular, if the following parameters are used,*

$$\varepsilon_T = \frac{\varepsilon(1-\gamma)}{16V_{\max}}, \quad \varepsilon_R = \frac{\varepsilon(1-\gamma)}{16}, \quad \varepsilon_P = \frac{\varepsilon(1-\gamma)}{24}, \quad \delta_T = \delta_R = \frac{\delta}{4},$$

*then the sample complexity of exploration of **KWIK-Rmax** is*

$$O\left(\frac{V_{\max}}{\varepsilon(1-\gamma)} \left(B_T(\varepsilon_T, \delta_T) + B_R(\varepsilon_R, \delta_R) + \ln \frac{1}{\delta}\right) \ln \frac{1}{\varepsilon(1-\gamma)}\right).$$

We omit the proof which is a generalization of that for **Rmax** (Theorem 6.1). Instead, we describe a few classes of KWIK-learnable MDPs and show how **KWIK-Rmax** unifies and extends previous PAC-MDP algorithms. Since transition

probabilities are usually more difficult to learn than rewards, we only describe how to KWIK-learn transition functions. Reward functions may be KWIK-learned using the same algorithmic building blocks. More examples are found in Li et al (2011).

Example 6.6. It is easy to see that **KWIK-Rmax** is a generalization of **Rmax**. In fact, **Rmax** may be interpreted as **KWIK-Rmax** that uses the a meta algorithm known as input partition to KWIK-learn a finite MDP with n states and m actions. This meta algorithm, denoted \mathbf{A}^* , runs nm instances of **dice-learning** indexed by state–actions. Whenever \mathbf{A}^* receives an input $x = (s, a) \in \mathbf{S} \times \mathbf{A}$, it queries the instance indexed by (s, a) with x , and makes the same prediction as the instance. If the prediction is \perp , \mathbf{A}^* will observe an observation (in this case, a next state from (s, a)) which it will pass to the corresponding **dice-learning** instance to allow it to learn. The accuracy requirement is satisfied with high probability since all valid predictions made by **dice-learning** are ε -accurate with high probability. Ignoring logarithmic factors, the KWIK bound of \mathbf{A}^* is the sum of KWIK bounds of individual **dice-learning** algorithms, *i.e.*, $\tilde{O}(n^2m/\varepsilon^2)$. Substituting this bound into Theorem 6.3 gives Theorem 6.2. \square

Example 6.7. In many robotics and adaptive control applications, the systems being manipulated have infinite state and action spaces, and their dynamics are governed by linear equations (Abbeel and Ng, 2005; Strehl and Littman, 2008b). Here, $\mathbf{S} \subseteq \mathbb{R}^{ns}$ and $\mathbf{A} \subseteq \mathbb{R}^{na}$ are the vector-valued state and action spaces, respectively. State transitions follow a multivariate normal distribution: given the current state–action (s, a) , the next state s' is randomly drawn from $\mathcal{N}(F\phi(s, a), \Sigma)$, where $\phi : \mathbb{R}^{ns+na} \rightarrow \mathbb{R}^n$ is a basis function satisfying $\|\phi(\cdot, \cdot)\| \leq 1$, $F \in \mathbb{R}^{ns \times n}$ is a matrix, $\Sigma \in \mathbb{R}^{ns \times ns}$ is a covariance matrix. We assume that ϕ and Σ are given, but F is unknown.

For such MDPs, each component of mean vector $\mathbf{E}[s']$ is a linear function of $\phi(s, a)$. Therefore, n_s instances of **noisy linear regression** may be used to KWIK-learn the n_s rows of F .⁵ If individual components of $\mathbf{E}[s']$ can be predicted accurately by the sub-algorithms, one can easily concatenate them to predict the entire vector $\mathbf{E}[s']$; otherwise, we can predict \perp and obtain data to allow the sub-algorithms to learn. This meta algorithm, known as **output combination**, allows one to KWIK-learn such linearly parameterized MDPs with a KWIK bound of $\tilde{O}(n_s^2n/\varepsilon^4)$. Therefore, the corresponding instantiation of **KWIK-Rmax** is PAC-MDP according to Theorem 6.3.

A related class of MDPs is the normal offset model motivated by robot navigation tasks (Brunskill et al, 2009), which may be viewed as a special case of the linear dynamics MDPs. \square

Example 6.8. Factored-state representations are compact representations of a special class of MDPs. Let $m = |\mathbf{A}|$ be the number of actions. Every state is a vector consisting of n components: $s = (s[1], s[2], \dots, s[n]) \in \mathbf{S}$. Each component $s[i]$ is called a *state variable* and takes values in a finite set \mathbf{S}_i . The whole state space \mathbf{S} is

⁵ One technical subtlety exists: one has to first “clip” a normal distribution into a distribution with bounded support since bounded noise is assumed in **noisy linear regression**.

thus $\mathbf{S}_1 \times \dots \times \mathbf{S}_n$. The transition function is assumed to be factored into a product of n transition functions, one for each state variable:

$$T(s,a,s') = \prod_{i=1}^n T_i(\mathbf{P}_i(s), a, s'[i]),$$

where $\mathbf{P}_i(s) \subseteq \{s[1], s[2], \dots, s[n]\}$, known as the i -th parent set, contains state variables that are *relevant* for defining the transition probabilities of $s'[i]$. Define two quantities: $D \stackrel{\text{def}}{=} \max_i |\mathbf{P}_i|$, the maximum size of parent sets, and $N \stackrel{\text{def}}{=} \max_i |\mathbf{S}_i|$, the maximum number of values a state variable can take. Therefore, although there are mN^{2n} transition probabilities in a factored-state MDP, the MDP is actually specified by no more than mnN^{D+1} free parameters. When $D \ll n$, such an MDP can not only be represented efficiently, but also be KWIK-learned with a small KWIK bound, as shown next.

When the parent sets \mathbf{P}_i are known *a priori*, output combination can be used to combine predictions for T_i , each of which can be KWIK-learned by an instance of input partition (over all possible mN^D state–actions) applied to dice learning (for multinomial distribution over N elements). This three-level KWIK algorithm provides an approach to learning the transition function of a factored-state MDP with the following KWIK bound (Li et al., 2011): $\tilde{O}(n^3 m D N^{D+1} / \varepsilon^2)$. This insight can be used to derive PAC-MDP algorithms for factored-state MDPs (Guestrin et al., 2002; Kearns and Koller, 1999; Strehl, 2007a).

In the more interesting case where the sets \mathbf{P}_i are unknown, the RL agent has to learn the *structure* of the factored-state representation. The problem of efficient exploration becomes very challenging when combined with the need for structure learning (Kearns and Koller, 1999). Fortunately, assuming knowledge of D , we can use the *noisy union* algorithm (Li et al., 2011) to KWIK-learn the set of possible structures, which is tractable as long as D is small. Combining *noisy union* with the three-level KWIK algorithm for the known structure case above not only simplifies an existing structure-learning algorithm (Strehl et al., 2007), but also facilitates development of more efficient ones (Diuk et al., 2009). \square

The examples above show the power of the KWIK model when it is combined with the generic PAC-MDP result in Theorem 6.3. Specifically, these examples show how to KWIK-learn various important classes of MDPs, each of which leads immediately to an instance of KWIK-Rmax that is PAC-MDP in those MDPs.

We conclude this section with two important open questions regarding the use of KWIK model for devising PAC-MDP algorithms:

- KWIK learning a hypothesis class becomes very challenging when the realizability assumption is violated, that is, when $h^* \notin \mathbf{H}$. In this case, some straightforward adaptation of the accuracy requirement in Definition 6.9 can make it impossible to KWIK-learn a hypothesis class, even if the class is KWIK-learnable in the realizable case (Li and Littman, 2010). Recently, an interesting effort has been made by Szita and Szepesvári (2011).

- When we discuss KWIK-learnable MDP classes, the *dimension* $\dim(M)$ for an MDP M is defined in a case-by-case manner. In general, it is unknown how to define complexity measure of hypothesis classes useful in the KWIK model.

6.6 Model-Free Approaches

The previous section discusses how to create model-based PAC-MDP RL algorithms systematically with help of KWIK algorithms for various families of MDPs. While model-based algorithms often have better sample complexity in practice, they usually maintain an explicit model of the unknown MDP and often have to repeatedly solve this approximate MDP (such as in KWIK-Rmax) to obtain a value function or policy for taking actions. Despite recent advances in approximate planning in MDPs, solving MDPs remains challenging in general.

In contrast, *model-free* algorithms directly learn optimal value functions, from which a greedy policy can be easily computed. These algorithms often have lower per-step computation complexity and thus are better suited for real-time applications, where computation resources for taking an action is limited. In finite MDPs, for example, both **Q-learning** and **Sarsa** require $O(\ln |A|)$ per-step time complexity, using a heap structure to maintain the state-action value estimates, and $\Theta(|S||A|)$ space complexity for storing those estimates. In contrast, solving a finite MDP requires $\Omega(|S||A|)$ space and time in general. However, these classic model-free algorithms do not specify a concrete exploration strategy, and in practice heuristics like ϵ -greedy are often used (Sutton and Barto, 1998). As shown in Example 6.1 (see also the examples by Strehl (2007b, Section 4.1)), these popular choices are not as reliable as PAC-MDP algorithms. Existing convergence rates often depend on quantities such as covering time (Even-Dar and Mansour, 2003) that may be exponentially large with a poor exploration strategy.

Delayed Q-learning (Strehl et al, 2006b) is the first model-free algorithm proved to be PAC-MDP. It generalizes earlier results for deterministic MDPs (Koenig and Simmons, 1996) and is representative for a few variants as well as some randomized versions of real-time dynamic programming (Strehl, 2007b). The version given in Algorithm 14 slightly improves and simplifies the original one (Strehl et al, 2006b), avoiding a somewhat unnatural exploration bonus.

The algorithm is called “delayed” because it waits until a state-action has been experienced m times before updating its Q-value, where m is an input parameter. When it updates the Q-value of a state-action, the update can be viewed as an average of the target values for the m most recently missed update opportunities. In this sense, the update rule of **delayed Q-learning** is like real-time dynamic programming (Barto et al, 1995) but it approximates the Bellman operator using samples.

We note that the batch update achieved by averaging m samples is equivalent to performing stochastic gradient descent using these samples with a sequence of learning rates: $1, \frac{1}{2}, \dots, \frac{1}{m}$, assuming updates on $Q(s,a)$ do not affect the sequence of samples. This observation justifies the name of the algorithm, emphasizing both the

```

0: Inputs:  $\mathbf{S}, \mathbf{A}, \gamma, m \in \mathbb{N}, \xi \in \mathbb{R}_+$ 
1: for all  $(s,a) \in \mathbf{S} \times \mathbf{A}$  do
2:    $Q(s,a) \leftarrow V_{\max}$  {optimistic initialization of state-action values}
3:    $U(s,a) \leftarrow 0$  {used for attempted updates of the value in  $(s,a)$ }
4:    $B(s,a) \leftarrow 0$  {beginning timestep of attempted update in  $(s,a)$ }
5:    $C(s,a) \leftarrow 0$  {counter for  $(s,a)$ }
6:    $L(s,a) \leftarrow \text{TRUE}$  {the learning flag}
7:    $t^* \leftarrow 0$  {timestamp of most recent state-action value change}
8: for  $t = 1, 2, 3, \dots$  do
9:   Observe the current state  $s_t$ , take action  $a_t \leftarrow \arg \max_{a \in \mathbf{A}} Q(s_t, a)$ , receive reward
     $r_t \in [0, 1]$ , and transition to a next state  $s_{t+1}$ .
10:  if  $B(s_t, a_t) \leq t^*$  then
11:     $L(s_t, a_t) \leftarrow \text{TRUE}$ 
12:  if  $L(s_t, a_t) = \text{TRUE}$  then
13:    if  $C(s_t, a_t) = 0$  then
14:       $B(s_t, a_t) \leftarrow t$ 
15:       $C(s_t, a_t) \leftarrow C(s_t, a_t) + 1$ 
16:       $U(s_t, a_t) \leftarrow U(s_t, a_t) + r_t + \gamma \max_{a \in \mathbf{A}} Q(s_{t+1}, a)$ 
17:    if  $C(s_t, a_t) = m$  then
18:       $q \leftarrow U(s_t, a_t)/m$ 
19:      if  $Q(s_t, a_t) - q \geq \xi$  {if Q-function changes significantly} then
20:         $Q(s_t, a_t) \leftarrow q$ 
21:         $t^* \leftarrow t$ 
22:      else if  $B(s_t, a_t) > t^*$  then
23:         $L(s_t, a_t) \leftarrow \text{FALSE}$ 
24:       $U(s_t, a_t) \leftarrow 0$ 
25:       $C(s_t, a_t) \leftarrow 0$ 

```

Algorithm 14. Delayed Q-learning

relation to Q-learning with the harmonic learning rates above and the batch nature of the updates.

To encourage exploration, delayed Q-learning uses the same optimism-in-the-face-of-uncertainty principle as many other algorithms like Rmax, MBIE and UCRL2. Specifically, its initial Q-function is an over-estimate of the true function; during execution, the successive value function estimates remain over-estimates with high probability as long as m is sufficiently large, as will be shown soon.

While the role of the learning flags may not be immediately obvious, they are critical to guarantee that the delayed update rule can happen only a finite number of times during an entire execution of delayed Q-learning. This fact will be useful when proving the following theorem for the algorithm's sample complexity.

Theorem 6.4. *In delayed Q-learning, if parameters are set by*

$$m = O\left(\frac{V_{\max}^2}{\varepsilon^2(1-\gamma)^2} \ln \frac{|\mathbf{S}| |\mathbf{A}|}{\varepsilon \delta (1-\gamma)}\right) \quad \text{and} \quad \xi = \frac{\varepsilon(1-\gamma)}{8},$$

then the algorithm is PAC-MDP with the following sample complexity:

$$O\left(\frac{|\mathbf{S}||\mathbf{A}|V_{\max}^4}{\varepsilon^4(1-\gamma)^4} \ln \frac{1}{\varepsilon(1-\gamma)} \ln \frac{|\mathbf{S}||\mathbf{A}|}{\varepsilon\delta(1-\gamma)}\right).$$

It is worth noting that the algorithm's sample complexity matches the lower bound of Li (2009) in terms of the number of states. While the sample complexity of exploration is worse than the recently proposed **MoRmax** (Szita and Szepesvári, 2010), its analysis is the first to make use of an interesting notion of known state–actions that is different from the one used by previous model-based algorithms like **Rmax**. Let Q_t be the value function estimate at timestep t , then the set of known state–actions are those with essentially nonnegative Bellman errors:

$$\mathbf{K}_t \stackrel{\text{def}}{=} \left\{ (s,a) \mid R(s,a) + \gamma \sum_{s' \in \mathbf{S}} T(s,a,s') \max_{a' \in \mathbf{A}} Q_t(s',a') - Q_t(s,a) \geq -\frac{\varepsilon(1-\gamma)}{4} \right\}.$$

A few definitions are useful in the proof. An *attempted update* of a state–action (s,a) is a timestep t for which (s,a) is experienced, $L(s,a) = \text{TRUE}$, and $C(s,a) = m$; in other words, attempted updates correspond to the timesteps in which Lines 18–25 in Algorithm 14 are executed. If, in addition, the condition in Line 19 is satisfied, the attempted update is called *successful* since the state–action value is changed; otherwise, it is called *unsuccessful*.

Proof. (sketch) For convenience, let $S = |\mathbf{S}|$ and $A = |\mathbf{A}|$. We first prove that there are only finitely many attempted updates in a whole run of the algorithm. Obviously, the number of successful updates is at most $\kappa \stackrel{\text{def}}{=} SAV_{\max}/\xi$ since the state–action value function is initialized to V_{\max} , the value function estimate is always non-negative, and every successful update decreases $Q(s,a)$ by at least ξ for some (s,a) . Now consider a fixed state–action (s,a) . Once (s,a) is experienced for the m -th time, an attempted update will occur. Suppose that an attempted update of (s,a) occurs during timestep t . Afterward, for another attempted update to occur during some later timestep $t' > t$, it must be the case that a successful update of some state–action (not necessarily (s,a)) has occurred on or after timestep t and before timestep t' . We have proved at most κ successful updates are possible and so there are at most $1 + \kappa$ attempted updates of (s,a) . Since there are SA state–actions, there can be at most $SA(1 + \kappa)$ total attempted updates. The bound on the number of attempted updates allows one to use a union bound to show that (Strehl et al, 2009, Lemma 22), with high probability, any attempted update on an $(s,a) \in \mathbf{K}_t$ will be successful using the specified value of m .

We are now ready to verify the three conditions in Theorem 6.1. The optimism condition is easiest to verify using mathematical induction. Delayed Q-learning uses optimistic initialization of the value function, so $Q_t(s,a) \geq Q^*(s,a)$ and thus $V_t(s) \geq V^*(s)$ for $t = 1$. Now, suppose $Q_t(s,a) \geq Q^*(s,a)$ and $V_t(s) \geq V^*(s)$ for all (s,a) . If some $Q(s,a)$ is updated, then Hoeffding's inequality together with the specified value for m ensures that the new value of $Q(s,a)$ is still optimistic (modulo a small gap of $O(\varepsilon)$ because of approximation error). Since there are only finitely

many attempted updates, we may apply the union bound so that $V_t(s) \geq V^*(s) - \varepsilon/4$ for all t , with high probability.

The accuracy condition can be verified using the definition of \mathbf{K}_t . By definition, the Bellman errors in known state–actions are at least $-\varepsilon(1 - \gamma)/4$. On the other hand, unknown state–actions have zero Bellman error, by Definition 6.7. Therefore, the well-known monotonicity property of Bellman operators implies that Q_t is uniformly greater than the optimal Q-function in the known state–action MDP $M_{\mathbf{K}_t}$, modulo a small gap of $O(\varepsilon)$. Thus the accuracy condition holds.

For the bounded-surprises condition, we first observe that the number of updates to $Q(\cdot, \cdot)$ is at most κ , as argued above. For the number of visits to unknown state–actions, it can be argued that, with high probability, an attempted update to the Q-value of an unknown state–action will be successful (using the specified value for m), and an unsuccessful update followed by $L(s, a) = \text{FALSE}$ indicates $(s, a) \in \mathbf{K}_t$. A formal argument requires more work but is along the same line of reasoning on the learning flags when we bounded the number of attempted updates above (Strehl et al, 2009, Lemma 25). \square

Delayed Q-learning may be combined with techniques like interval estimation to gain further improvement (Strehl, 2007b). Although PAC-MDP algorithms like **delayed Q-learning** exist for finite MDPs, extending the analysis to general MDPs turns out very challenging, unlike the case for the model-based **Rmax** algorithm. Part of the reason is the difficulty in analyzing model-free algorithms that often use bootstrapping when updating value functions. While the learning target in model-based algorithms is a fixed object (namely, the MDP model), the learning target in a model-free algorithm is often not fixed. In a recent work (Li and Littman, 2010), a finite-horizon reinforcement-learning problem in general MDPs is reduced to a series of KWIK regression problems so that we can obtain a model-free PAC-MDP RL algorithm as long as the individual KWIK regression problems are solvable. However, it remains open how to devise a KWIK-based model-free RL algorithm in discounted problems without first converting it into a finite-horizon one.

6.7 Concluding Remarks

Exploration is a fundamental problem in reinforcement learning. Efficient exploration implies that an RL agent is able to experiment with an unknown environment to figure out a near-optimal policy quickly. On the other hand, inefficient exploration often yields slow convergence (or even non-convergence) to near-optimal policies. This chapter surveys provably efficient exploration schemes in the literature. While we focus on one such formal framework known as PAC-MDP, we also briefly discuss other alternatives, including Bayesian exploration and regret minimization, and compare them to PAC-MDP.

The PAC-MDP framework provides a rigorous framework for studying the exploration problem. It also provides a lot of flexibility. Both model-based and model-free PAC-MDP algorithms can be developed. More importantly, with help from the

KWIK model, PAC-MDP algorithms can be systematically developed for many useful MDP classes beyond finite MDPs. Examples include various rich classes of problems like factored-state MDPs (Guestrin et al, 2002; Kearns and Koller, 1999; Strehl et al, 2007; Diuk et al, 2009), continuous-state MDPs (Strehl and Littman, 2008b; Brunskill et al, 2009), and relational MDPs (Walsh, 2010). Furthermore, ideas from the KWIK model can also be combined with apprenticeship learning, resulting in RL systems that can explore more efficiently with the help of a teacher (Walsh et al, 2010b; Sayedi et al, 2011; Walsh et al, 2012).

Although the worst-case sample complexity bounds may be conservative, the principled algorithms and analyses have proved useful for guiding development of more practical exploration schemes. A number of novel algorithms are motivated and work well in practice (Jong and Stone, 2007; Li et al, 2009; Nouri and Littman, 2009, 2010), all of which incorporate various notion of knowleness in guiding exploration. Several PAC-MDP algorithms are also competitive in non-trivial applications like robotics (Brunskill et al, 2009) and computer games (Diuk et al, 2008).

Acknowledgements. The author would like to thank John Langford, Michael Littman, Alex Strehl, Tom Walsh, and Eric Wiewiora for significant contributions to the development of the KWIK model and sample complexity analysis of a number of PAC-MDP algorithms. The reviewers and editors of the chapter as well as Michael Littman and Tom Walsh have provided valuable comments that improves the content and presentation of the article in a number of substantial ways.

References

- Abbeel, P., Ng, A.Y.: Exploration and apprenticeship learning in reinforcement learning. In: Proceedings of the Twenty-Second International Conference on Machine Learning (ICML-2005), pp. 1–8 (2005)
- Asmuth, J., Li, L., Littman, M.L., Nouri, A., Wingate, D.: A Bayesian sampling approach to exploration in reinforcement learning. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-2009), pp. 19–26 (2009)
- Bartlett, P.L., Tewari, A.: REGAL: A regularization based algorithm for reinforcement learning in weakly communicating MDPs. In: Proceedings of the Twenty-Fifth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2009), pp. 35–42 (2009)
- Barto, A.G., Bradtko, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1-2), 81–138 (1995)
- Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Athena Scientific (1996)
- Brafman, R.I., Tennenholtz, M.: R-max—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3, 213–231 (2002)
- Brunskill, E., Leffler, B.R., Li, L., Littman, M.L., Roy, N.: Provably efficient learning with typed parametric models. *Journal of Machine Learning Research* 10, 1955–1988 (2009)
- Burnetas, A.N., Katehakis, M.N.: Optimal adaptive policies for Markov decision processes. *Mathematics of Operations Research* 22(1), 222–255 (1997)

- Dearden, R., Friedman, N., Andre, D.: Model based Bayesian exploration. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI-1999), pp. 150–159 (1999)
- Diuk, C., Cohen, A., Littman, M.L.: An object-oriented representation for efficient reinforcement learning. In: Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-2008), pp. 240–247 (2008)
- Diuk, C., Li, L., Leffler, B.R.: The adaptive k -meteorologists problem and its application to structure discovery and feature selection in reinforcement learning. In: Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML-2009), pp. 249–256 (2009)
- Duff, M.O.: Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes. PhD thesis, University of Massachusetts, Amherst, MA (2002)
- Even-Dar, E., Mansour, Y.: Learning rates for Q-learning. *Journal of Machine Learning Research* 5, 1–25 (2003)
- Even-Dar, E., Mannor, S., Mansour, Y.: Multi-Armed Bandit and Markov Decision Processes. In: Kivinen, J., Sloan, R.H. (eds.) COLT 2002. LNCS (LNAI), vol. 2375, pp. 255–270. Springer, Heidelberg (2002)
- Fiechter, C.N.: Efficient reinforcement learning. In: Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory (COLT-1994), pp. 88–97 (1994)
- Fiechter, C.N.: Expected mistake bound model for on-line reinforcement learning. In: Proceedings of the Fourteenth International Conference on Machine Learning (ICML-1997), pp. 116–124 (1997)
- Guestrin, C., Patrascu, R., Schuurmans, D.: Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. In: Proceedings of the Nineteenth International Conference on Machine Learning (ICML-2002), pp. 235–242 (2002)
- Jaakkola, T., Jordan, M.I., Singh, S.P.: On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation* 6(6), 1185–1201 (1994)
- Jaksch, T., Ortner, R., Auer, P.: Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research* 11, 1563–1600 (2010)
- Jong, N.K., Stone, P.: Model-based function approximation in reinforcement learning. In: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2007), pp. 670–677 (2007)
- Kaelbling, L.P.: Learning in Embedded Systems. MIT Press, Cambridge (1993)
- Kakade, S.: On the sample complexity of reinforcement learning. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, UK (2003)
- Kakade, S., Kearns, M.J., Langford, J.: Exploration in metric state spaces. In: Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), pp. 306–312 (2003)
- Kearns, M.J., Koller, D.: Efficient reinforcement learning in factored MDPs. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-1999), pp. 740–747 (1999)
- Kearns, M.J., Singh, S.P.: Finite-sample convergence rates for Q-learning and indirect algorithms. In: Advances in Neural Information Processing Systems (NIPS-1998), vol. 11, pp. 996–1002 (1998)
- Kearns, M.J., Singh, S.P.: Near-optimal reinforcement learning in polynomial time. *Machine Learning* 49(2-3), 209–232 (2002)
- Kearns, M.J., Mansour, Y., Ng, A.Y.: A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning* 49(2-3), 193–208 (2002)

- Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
- Koenig, S., Simmons, R.G.: The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning* 22(1-3), 227–250 (1996)
- Kolter, J.Z., Ng, A.Y.: Near Bayesian exploration in polynomial time. In: Proceedings of the Twenty-Sixth International Conference on Machine Learning (ICML-2009), pp. 513–520 (2009)
- Li, L.: A unifying framework for computational reinforcement learning theory. PhD thesis, Rutgers University, New Brunswick, NJ (2009)
- Li, L., Littman, M.L.: Reducing reinforcement learning to KWIK online regression. *Annals of Mathematics and Artificial Intelligence* 58(3-4), 217–237 (2010)
- Li, L., Littman, M.L., Mansley, C.R.: Online exploration in least-squares policy iteration. In: Proceedings of the Eighteenth International Conference on Agents and Multiagent Systems (AAMAS-2009), pp. 733–739 (2009)
- Li, L., Littman, M.L., Walsh, T.J., Strehl, A.L.: Knows what it knows: A framework for self-aware learning. *Machine Learning* 82(3), 399–443 (2011)
- Littlestone, N.: Learning quickly when irrelevant attributes abound: A new linear-threshold algorithms. *Machine Learning* 2(4), 285–318 (1987)
- Meuleau, N., Bourgine, P.: Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning* 35(2), 117–154 (1999)
- Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13(1), 103–130 (1993)
- Neu, G., György, A., Szepesvári, C., Antos, A.: Online Markov decision processes under bandit feedback. In: Advances in Neural Information Processing Systems 23 (NIPS-2010), pp. 1804–1812 (2011)
- Ng, A.Y., Harada, D., Russell, S.J.: Policy invariance under reward transformations: Theory and application to reward shaping. In: Proceedings of the Sixteenth International Conference on Machine Learning (ICML-1999), pp. 278–287 (1999)
- Nouri, A., Littman, M.L.: Multi-resolution exploration in continuous spaces. In: Advances in Neural Information Processing Systems 21 (NIPS-2008), pp. 1209–1216 (2009)
- Nouri, A., Littman, M.L.: Dimension reduction and its application to model-based exploration in continuous spaces. *Machine Learning* 81(1), 85–98 (2010)
- Poupart, P., Vlassis, N., Hoey, J., Regan, K.: An analytic solution to discrete Bayesian reinforcement learning. In: Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-2006), pp. 697–704 (2006)
- Puterman, M.L.: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, New York (1994)
- Ratitch, B., Precup, D.: Using MDP Characteristics to Guide Exploration in Reinforcement Learning. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) ECML 2003. LNCS (LNAI), vol. 2837, pp. 313–324. Springer, Heidelberg (2003)
- Robbins, H.: Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society* 58(5), 527–535 (1952)
- Sayedi, A., Zadimoghaddam, M., Blum, A.: Trading off mistakes and don't-know predictions. In: Advances in Neural Information Processing Systems 23 (NIPS-2010), pp. 2092–2100 (2011)
- Singh, S.P., Jaakkola, T., Littman, M.L., Szepesvári, C.: Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning* 38(3), 287–308 (2000)

- Strehl, A.L.: Model-based reinforcement learning in factored-state MDPs. In: Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, pp. 103–110 (2007a)
- Strehl, A.L.: Probably approximately correct (PAC) exploration in reinforcement learning. PhD thesis, Rutgers University, New Brunswick, NJ (2007b)
- Strehl, A.L., Littman, M.L.: An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences* 74(8), 1309–1331 (2008a)
- Strehl, A.L., Littman, M.L.: Online linear regression and its application to model-based reinforcement learning. In: Advances in Neural Information Processing Systems 20 (NIPS-2007), pp. 1417–1424 (2008b)
- Strehl, A.L., Li, L., Littman, M.L.: Incremental model-based learners with formal learning-time guarantees. In: Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (UAI-2006), pp. 485–493 (2006a)
- Strehl, A.L., Li, L., Wiewiora, E., Langford, J., Littman, M.L.: PAC model-free reinforcement learning. In: Proceedings of the Twenty-Third International Conference on Machine Learning (ICML-2006), pp. 881–888 (2006b)
- Strehl, A.L., Diuk, C., Littman, M.L.: Efficient structure learning in factored-state MDPs. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-2007), pp. 645–650 (2007)
- Strehl, A.L., Li, L., Littman, M.L.: Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research* 10, 2413–2444 (2009)
- Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
- Szita, I., Lőrincz, A.: The many faces of optimism: A unifying approach. In: Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-2008), pp. 1048–1055 (2008)
- Szita, I., Szepesvári, C.: Model-based reinforcement learning with nearly tight exploration complexity bounds. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML-2010), pp. 1031–1038 (2010)
- Szita, I., Szepesvári, C.: Agnostic KWIK learning and efficient approximate reinforcement learning. In: Proceedings of the Twenty-Fourth Annual Conference on Learning Theory, COLT-2011 (2011)
- Tewari, A., Bartlett, P.L.: Optimistic linear programming gives logarithmic regret for irreducible MDPs. In: Advances in Neural Information Processing Systems 20 (NIPS-2007), pp. 1505–1512 (2008)
- Thrun, S.: The role of exploration in learning control. In: White, D.A., Sofge, D.A. (eds.) *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, pp. 527–559. Van Nostrand Reinhold (1992)
- Valiant, L.G.: A theory of the learnable. *Communications of the ACM* 27(11), 1134–1142 (1984)
- Walsh, T.J.: Efficient learning of relational models for sequential decision making. PhD thesis, Rutgers University, New Brunswick, NJ (2010)
- Walsh, T.J., Szita, I., Diuk, C., Littman, M.L.: Exploring compact reinforcement-learning representations with linear regression. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-2009), pp. 591–598 (2009); corrected version as Technical Report DCS-tr-660, Department of Computer Science, Rutgers University
- Walsh, T.J., Goschin, S., Littman, M.L.: Integrating sample-based planning and model-based reinforcement learning. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-2010), pp. 612–617 (2010a)

- Walsh, T.J., Subramanian, K., Littman, M.L., Diuk, C.: Generalizing apprenticeship learning across hypothesis classes. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML-2010), pp. 1119–1126 (2010b)
- Walsh, T.J., Hewlett, D., Morrison, C.T.: Blending autonomous and apprenticeship learning. In: Advances in Neural Information Processing Systems 24, NIPS-2011 (2012)
- Watkins, C.J., Dayan, P.: *Q*-learning. *Machine Learning* 8, 279–292 (1992)
- Whitehead, S.D.: Complexity and cooperation in Q-learning. In: Proceedings of the Eighth International Workshop on Machine Learning (ICML-1991), pp. 363–367 (1991)
- Wiering, M., Schmidhuber, J.: Efficient model-based exploration. In: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 5 (SAB-1998), pp. 223–228 (1998)

Part III
Constructive-Representational Directions

Chapter 7

Reinforcement Learning in Continuous State and Action Spaces

Hado van Hasselt

Abstract. Many traditional reinforcement-learning algorithms have been designed for problems with small finite state and action spaces. Learning in such discrete problems can be difficult, due to noise and delayed reinforcements. However, many real-world problems have continuous state or action spaces, which can make learning a good decision policy even more involved. In this chapter we discuss how to automatically find good decision policies in continuous domains. Because analytically computing a good policy from a continuous model can be infeasible, in this chapter we mainly focus on methods that explicitly update a representation of a value function, a policy or both. We discuss considerations in choosing an appropriate representation for these functions and discuss gradient-based and gradient-free ways to update the parameters. We show how to apply these methods to reinforcement-learning problems and discuss many specific algorithms. Amongst others, we cover gradient-based temporal-difference learning, evolutionary strategies, policy-gradient algorithms and (natural) actor-critic methods. We discuss the advantages of different approaches and compare the performance of a state-of-the-art actor-critic method and a state-of-the-art evolutionary strategy empirically.

7.1 Introduction

In this chapter, we consider the problem of sequential decision making in continuous domains with delayed reward signals. The full problem requires an algorithm to learn how to choose actions from an infinitely large action space to optimize a noisy delayed cumulative reward signal in an infinitely large state space, where even the outcome of a single action can be stochastic. Desirable properties of such an algorithm include applicability in many different instantiations of the general problem,

Hado van Hasselt

Centrum Wiskunde en Informatica (CWI, Center for Mathematics and Computer Science)
Amsterdam, The Netherlands
e-mail: H.van.Hasselt@cwi.nl

computational efficiency such that it can be used in real-time and sample efficiency such that it can learn good action-selection policies with limited experience.

Because of the complexity of the full reinforcement-learning problem in continuous spaces, many traditional reinforcement-learning methods have been designed for Markov decision processes (MDPs) with small finite state and action spaces. However, many problems inherently have large or continuous domains. In this chapter, we discuss how to use reinforcement learning to learn good action-selection policies in MDPs with continuous state spaces and discrete action spaces and in MDPs where the state and action spaces are both continuous.

Throughout this chapter, we assume that a model of the environment is not known. If a model is available, one can use dynamic programming (Bellman, 1957; Howard, 1960; Puterman, 1994; Sutton and Barto, 1998; Bertsekas, 2005, 2007), or one can sample from the model and use one of the reinforcement-learning algorithms we discuss below. We focus mainly on the problem of *control*, which means we want to find action-selection policies that yield high returns, as opposed to the problem of *prediction*, which aims to estimate the value of a given policy.

For general introductions to reinforcement learning from varying perspectives, we refer to the books by Bertsekas and Tsitsiklis (1996) and Sutton and Barto (1998) and the more recent books by Bertsekas (2007), Powell (2007), Szepesvári (2010) and Buşoniu et al (2010). Whenever we refer to a chapter, it is implied to be the relevant chapter from the same volume as this chapter.

In the remainder of this introduction, we describe the structure of MDPs in continuous domains and discuss three general methodologies to find good policies in such MDPs. We discuss function approximation techniques to deal with large or continuous spaces in Section 7.2. We apply these techniques to reinforcement learning in Section 7.3, where we discuss the current state of knowledge for reinforcement learning in continuous domains. This includes discussions on temporal differences, policy gradients, actor-critic algorithms and evolutionary strategies. Section 7.4 shows the results of an experiment, comparing an actor-critic method to an evolutionary strategy on a double-pole cart pole. Section 7.5 concludes the chapter.

7.1.1 *Markov Decision Processes in Continuous Spaces*

A Markov decision process (MDP) is a tuple (S, A, T, R, γ) . In this chapter, the state space S is generally an infinitely large bounded set. More specifically, we assume the state space is a subset of a possibly multi-dimensional Euclidean space, such that $S \subseteq \mathbb{R}^{D_S}$, where $D_S \in \mathbb{N}$ is the dimension of the state space. The action space is discrete or continuous and in the latter case we assume $A \subseteq \mathbb{R}^{D_A}$, where $D_A \in \mathbb{N}$ is the dimension of the action space.¹ We consider two variants: MDPs with continuous states and discrete actions and MDPs where both the states and actions

¹ In general, the action space is more accurately represented with a function that maps a state into a continuous set, such that $A(s) \subseteq \mathbb{R}^{D_A}$. We ignore this subtlety for conciseness.

Table 7.1 Symbols used in this chapter. All vectors are column vectors

$D_X \in \{1, 2, \dots\}$	dimension of space X
$S \subseteq \mathbb{R}^{D_S}$	state space
$A \subseteq \mathbb{R}^{D_A}$	action space
$T : S \times A \times S \rightarrow [0, 1]$	state-transition function
$R : S \times A \times S \rightarrow \mathbb{R}$	expected-reward function
$\gamma \in [0, 1]$	discount factor
$V : S \rightarrow \mathbb{R}$	state value function
$Q : S \times A \rightarrow \mathbb{R}$	state-action value function
$\pi : S \times A \rightarrow [0, 1]$	action-selection policy
$\alpha \in \mathbb{R}, \beta \in \mathbb{R}$	step-size parameters (may depend on state and action)
$t \in \mathbb{N}$	time step
$k \in \mathbb{N}$	episode
$\Phi \subseteq \mathbb{R}^{D_\Phi}$	feature space
$\phi : S \rightarrow \Phi$	feature-extraction function
$\Theta \subseteq \mathbb{R}^{D_\Theta}$	parameter space for value functions
$\theta \in \Theta$	parameter vector for a value function
$\Psi \subseteq \mathbb{R}^{D_\Psi}$	parameter space for policies
$\psi \in \Psi$	parameter vector for a policy
$e \in \mathbb{R}^{D_\Theta}$	eligibility trace vector
$\ \mathbf{x}\ = \sum_{i=0}^n x[i]^2$	quadratic norm of vector $\mathbf{x} = \{x[0], \dots, x[n]\}$
$\ f\ = \int_{x \in X} (f(x))^2 dx$	quadratic norm of function $f : X \rightarrow \mathbb{R}$
$\ f\ _w = \int_{x \in X} w(x) (f(x))^2 dx$	quadratic weighted norm of function $f : X \rightarrow \mathbb{R}$

are continuous. Often, when we write ‘continuous’ the results hold for ‘large finite’ spaces as well. The notation used in this chapter is summarized in Table 7.1.

The transition function $T(s, a, s')$ gives the probability of a transition to s' when action a is performed in s . When the state space is continuous, we can assume the transition function specifies a probability density function (PDF), such that

$$\int_{S'} T(s, a, s') ds' = P(s_{t+1} \in S' | s_t = s \text{ and } a_t = a)$$

denotes the probability that action a in state s results in a transition to a state in the region $S' \subseteq S$. It is often more intuitive to describe the transitions through a function that describes the system dynamics, such that

$$s_{t+1} = T(s_t, a_t) + \omega_T(s_t, a_t) ,$$

where $T : S \times A \rightarrow S$ is a deterministic transition function that returns the expected next state for a given state-action pair and $\omega_T(s, a)$ is a zero-mean noise vector with the same size as the state vector. For example, s_{t+1} could be sampled from a Gaussian distribution centered at $T(s_t, a_t)$. The reward function gives the expected reward for any two states and an action. The actual reward can contain noise:

$$r_{t+1} = R(s_t, a_t, s_{t+1}) + \omega_R(s_t, a_t, s_{t+1}) ,$$

where $\omega_R(s,a,s')$ is a real-valued zero-mean noise term. If ω_R and the components of ω_T are not uniformly zero at all time steps, the MDP is called stochastic. Otherwise it is deterministic. If T or R is time-dependent, the MDP is non-stationary. In this chapter, we assume stationary MDPs. Since it is commonly assumed that S , A and γ are known, when we refer to a *model* in this chapter we usually mean (approximations of) T and R .

When only the state space is continuous, the action-selection policy is represented by a state dependent probability mass function $\pi : S \times A \rightarrow [0,1]$, such that

$$\pi(s,a) = P(a_t = a | s_t = s) \quad \text{and} \quad \sum_{a \in A} \pi(s,a) = 1 .$$

When the action space is also continuous, $\pi(s)$ represents a PDF on the action space.

The goal of prediction is to find the value of the *expected future discounted reward* for a given policy. The goal of control is to optimize this value by finding an optimal policy. It is useful to define the following operators $B^\pi : \mathcal{V} \rightarrow \mathcal{V}$ and $B^* : \mathcal{V} \rightarrow \mathcal{V}$, where \mathcal{V} is the space of all value functions:²

$$\begin{aligned} (B^\pi V)(s) &= \int_A \pi(s,a) \int_S T(s,a,s') (R(s,a,s') + \gamma V(s')) ds' da , \\ (B^* V)(s) &= \max_a \int_S T(s,a,s') (R(s,a,s') + \gamma V(s')) ds' , \end{aligned} \quad (7.1)$$

In continuous MDPs, the values of a given policy and the optimal value can then be expressed with the Bellman equations $V^\pi = B^\pi V^\pi$ and $V^* = B^* V^*$. Here $V^\pi(s)$ is the value of performing policy π starting from state s and $V^*(s) = \max_\pi V^\pi(s)$ is the value of the best possible policy. If the action space is finite, the outer integral in equation (7.1) should be replaced with a summation. In this chapter, we mainly consider discounted MDPs, which means that $\gamma \in (0,1)$.

For control with finite action spaces, action values are often used. The optimal action value for continuous state spaces is given by the Bellman equation

$$Q^*(s,a) = \int_S T(s,a,s') \left(R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right) ds' . \quad (7.2)$$

The idea is that when Q^* is approximated by Q with sufficient accuracy, we get a good policy by selecting the argument a that maximizes $Q(s,a)$ in each state s . Unfortunately, when the action space is continuous both this selection and the max operator in equation (7.2) may require finding the solution for a non-trivial optimization problem. We discuss algorithms to deal with continuous actions in Section 7.3. First, we discuss three general ways to learn good policies in continuous MDPs.

² In the literature, these operators are more commonly denoted T^π and T^* (e.g., Szepesvári, 2010), but since we use T to denote the transition function, we choose to use B .

7.1.2 Methodologies to Solve a Continuous MDP

In the problem of control, the aim is an approximation of the optimal policy. The optimal policy depends on the optimal value, which in turn depends on the model of the MDP. In terms of equation (7.2), the optimal policy is the policy π^* that maximizes Q^* for each state: $\sum_a \pi^*(s,a)Q^*(s,a) = \max_a Q^*(s,a)$. This means that rather than trying to estimate π^* directly, we can try to estimate Q^* , or we can even estimate T and R to construct Q^* and π^* when needed. These observations lead to the following three general methodologies that differ in which part of the solution is explicitly approximated. These methodologies are not mutually exclusive and we will discuss algorithms that use combinations of these approaches.

Model Approximation. Model-approximation algorithms approximate the MDP and compute the desired policy on this approximate MDP. Since S , A and γ are assumed to be known, this amounts to learning an approximation for the functions T and R .³ Because of the Markov property, these functions only depend on local data. The problem of estimating these functions then translates to a fairly standard supervised learning problem. For instance, one can use Bayesian methods (Dearden et al, 1998, 1999; Strens, 2000; Poupart et al, 2006) to estimate the required model. Learning the model may not be trivial, but in general it is easier than learning the value of a policy or optimizing the policy directly. For a recent survey on model-learning algorithms, see Nguyen-Tuong and Peters (2011).

An approximate model can be used to compute a value function. This can be done iteratively, for instance using *value iteration* or *policy iteration* (Bellman, 1957; Howard, 1960; Puterman and Shin, 1978; Puterman, 1994). The major drawback of model-based algorithms in continuous-state MDPs is that even if a model is known, in general one cannot easily extract a good policy from the model for all possible states. For instance, value iteration uses an inner loop over the whole state space, which is impossible if this space is infinitely large. Alternatively, a learned model can be used to generate sample runs. These samples can then be used to estimate a value function, or to improve the policy, using one of the methods outlined below. However, if the accuracy of the model is debatable, the resulting policy may not be better than a policy that is based directly on the samples that were used to construct the approximate model. In some cases, value iteration can be feasible, for instance because $T(s,a,s')$ is non-zero for only a small number of states s' . Even so, it may be easier to approximate the value directly than to infer the values from an approximate model. For reasons of space, we will not consider model approximation further.

Value Approximation. In this second methodology, the samples are used to approximate V^* or Q^* directly. Many reinforcement-learning algorithms fall into this category. We discuss value-approximation algorithms in Section 7.3.1.

³ In engineering, the reward function is usually considered to be known. Unfortunately, this does not make things much easier, since the transition function is usually harder to estimate anyway.

Policy Approximation. Value-approximation algorithms parametrize the policy indirectly by estimating state or action values from which a policy can be inferred. Policy-approximation algorithms store a policy directly and try update this policy to approximate the optimal policy. Algorithms that only store a policy, and not a value function, are often called *direct policy-search* (Ng et al, 1999) or *actor-only* algorithms (Konda and Tsitsiklis, 2003). Algorithms that store both a policy and a value function are commonly known as *actor-critic* methods (Barto et al, 1983; Sutton, 1984; Sutton and Barto, 1998; Konda, 2002; Konda and Tsitsiklis, 2003). We will discuss examples of both these approaches. Using this terminology, value-based algorithms that do not store an explicit policy can be considered *critic-only* algorithms. Policy-approximation algorithms are discussed in Section 7.3.2.

7.2 Function Approximation

Before we discuss algorithms to update approximations of value functions or policies, we discuss general ways to store and update an approximate function. General methods to learn a function from data are the topic of active research in the field of machine learning. For general discussions, see for instance the books by Vapnik (1995), Mitchell (1996) and Bishop (2006).

In Sections 7.2.1 and 7.2.2 we discuss linear and non-linear function approximation. In both cases, the values of the approximate function are determined by a set of tunable parameters. In Section 7.2.3 we discuss *gradient-based* and *gradient-free* methods to update these parameters. Both approaches have often been used in reinforcement learning with considerable success (Sutton, 1988; Werbos, 1989b,a, 1990; Whitley et al, 1993; Tesauro, 1994, 1995; Moriarty and Miikkulainen, 1996; Moriarty et al, 1999; Whiteson and Stone, 2006; Wierstra et al, 2008; Rückstieß et al, 2010). Because of space limitations, we will not discuss non-parametric approaches, such as kernel-based methods (see, e.g., Ormoneit and Sen, 2002; Powell, 2007; Buşoniu et al, 2010).

In this section, we mostly limit ourselves to the general functional form of the approximators and general methods to update the parameters. In order to apply these methods to reinforcement learning, there are a number of design considerations. For instance, we have to decide how to measure how accurate the approximation is. We discuss how to apply these methods to reinforcement learning in Section 7.3.

In supervised learning, a labeled data set is given that contains a number of inputs with the intended outputs for these inputs. One can then answer statistical questions about the process that spawned the data, such as the value of the function that generated the data at unseen inputs. In value-based reinforcement learning, targets may depend on an adapting policy or on adapting values of states. Therefore, targets may change during training and not all methods from supervised learning are directly applicable to reinforcement learning. Nonetheless, many of the same techniques can successfully be applied to the reinforcement learning setting, as long as one is careful about the inherent properties of learning in an MDP. First, we discuss some

issues on the choice of approximator. This discussion is split into a part on linear function approximation and one on non-linear function approximation.

7.2.1 Linear Function Approximation

We assume some feature-extraction function $\phi : S \rightarrow \Phi$ is given that maps states into features in the feature space Φ . We assume $\Phi \subseteq \mathbb{R}^{D_\Phi}$ where D_Φ is the dimension of the feature space. Often $D_\Phi < D_S$, which means that the feature vector of a state is smaller than the full representation of the state. This need not hold in general. A discussion about the choice of good features falls outside the scope of this chapter, but see for instance Buşoniu et al (2010) for some considerations.

A linear function is a simple parametric function that depends on the feature vector. For instance, consider a value-approximating algorithm where the value function is approximated by

$$V_t(s) = \theta_t^T \phi(s) . \quad (7.3)$$

In equation (7.3) and in the rest of this chapter, $\theta_t \in \Theta$ denotes the adaptable parameter vector at time t and $\phi(s) \in \Phi$ is the feature vector of state s . Since the function in equation (7.3) is linear in the parameters, we refer to it as a linear function approximator. Note that it may be non-linear in the state variables, depending on the feature extraction. In this section, the dimension D_Θ of the parameter space is equal to the dimension of the feature space D_Φ . This does not necessarily hold for other types of function approximation.

Linear function approximators are useful since they are better understood than non-linear function approximators. Applied to reinforcement learning, this has led to a number of convergence guarantees, under various additional assumptions (Sutton, 1984, 1988; Dayan, 1992; Peng, 1993; Dayan and Sejnowski, 1994; Bertsekas and Tsitsiklis, 1996; Tsitsiklis and Van Roy, 1997). From a practical point of view, linear approximators are useful because they are simple to implement and fast to compute.

Many problems have large state spaces in which each state can be represented efficiently with a feature vector of limited size. For instance, the double pole cart pole problem that we consider later in this chapter has continuous state variables, and therefore an infinitely large state space. Yet, every state can be represented with a vector with six elements. This means that we would need a table of infinite size, but can suffice with a parameter vector with just six elements if we use (7.3) with the state variables as features.

This reduction of tunable parameters of the value function comes at a cost. It is obvious that not every possible value function can be represented as a linear combination of the features of the problem. Therefore, our solution is limited to the set of value functions that can be represented with the chosen functional form. If one does not know beforehand what useful features are for a given problem, it can be beneficial to use non-linear function approximation, which we discuss in Section 7.2.2.

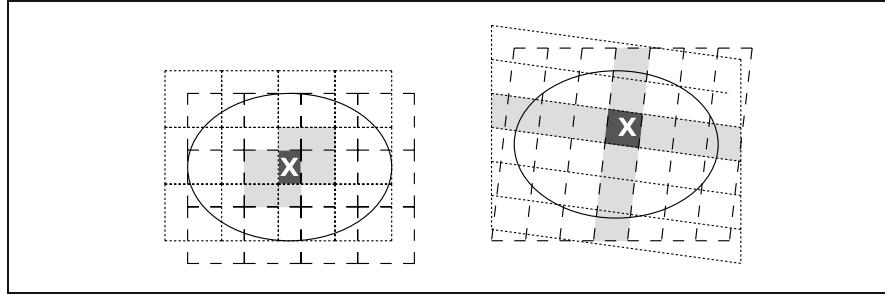


Fig. 7.1 An elliptical state space is discretized by tile coding with two tilings. For a state located at the X, the two active tiles are shown in light grey. The overlap of these active features is shown in dark grey. On the left, each tiling contains 12 tiles. The feature vector contains 24 elements and 35 different combinations of active features can be encountered in the elliptical state space. On the right, the feature vector contains 13 elements and 34 combinations of active features can be encountered, although some combinations correspond to very small parts of the ellipse.

7.2.1.1 Discretizing the State Space: Tile Coding

A common method to find features for a linear function approximator divides the continuous state space into separate segments and attaches one feature to each segment. A feature is active (i.e., equal to one) if the relevant state falls into the corresponding segment. Otherwise, it is inactive (i.e., equal to zero).

An example of such a discretizing method that is often used in reinforcement learning is *tile coding* (Watkins, 1989; Lin and Kim, 1991; Sutton, 1996; Santamaria et al, 1997; Sutton and Barto, 1998), which is based on the Cerebellar Model Articulation Controller (CMAC) structure proposed by Albus (1971, 1975). In tile coding, the state space is divided into a number of disjoint sets. These sets are commonly called tiles in this context. For instance, one could define N hypercubes such that each hypercube H_n is defined by a Cartesian product $H_n = [x_{n,1}, y_{n,1}] \times \dots \times [x_{n,D_S}, y_{n,D_S}]$, where $x_{n,d}$ is the lower bound of hypercube H_n in state dimension d and $y_{n,d}$ is the corresponding upper bound. Then, a feature $\phi_n(s) \in \phi(s)$ corresponding to H_n is equal to one when $s \in H_n$ and zero otherwise.

The idea behind tile coding is to use multiple non-overlapping tilings. If a single tiling contains N tiles, one could use M such tilings to obtain a feature vector of dimension $D_\phi = MN$. In each state, precisely M of these features are then equal to one, while the others are equal to zero. An example with $M = 2$ tilings and $D_\phi = 24$ features is shown on the left in Figure 7.1. The tilings do not have to be homogeneous. The right picture in Figure 7.1 shows a non-homogeneous example with $M = 2$ tilings and $D_\phi = 13$ features.

When M features are active for each state, up to $\binom{D_\phi}{M}$ different situations can theoretically be represented with D_ϕ features. This contrasts with the naive approach where only one feature is active for each state, which would only be able to

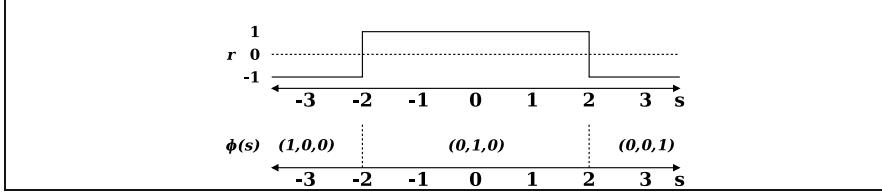


Fig. 7.2 A reward function and feature mapping. The reward is Markov for the features. If $s_{t+1} = s_t + a_t$ with $a_t \in \{-2, 2\}$, the feature-transition function is not Markov. This makes it impossible to determine an optimal policy.

represent D_ϕ different situations with the same number of features.⁴ In practice, the upper bound of $\binom{D_\phi}{M}$ will rarely be obtained, since many combinations of active features will not be possible. In both examples in Figure 7.1, the number of different possible feature vectors is indeed larger than the length of the feature vector and smaller than the theoretical upper bound: $24 < 35 < \binom{24}{2} = 276$ and $13 < 34 < \binom{13}{2} = 78$.

7.2.1.2 Issues with Discretization

One potential problem with discretizing methods such as tile coding is that the resulting function that maps states into features is not injective. In other words, $\phi(s) = \phi(s')$ does not imply that $s = s'$. This means that the resulting feature-space MDP is partially observable and one should consider using an algorithm that is explicitly designed to work on partially observable MDPs (POMDPs). For more on POMDPs, see Chapter 12. In practice, many good results have been obtained with tile coding, but the discretization and the resulting loss of the Markov property imply that most convergence proofs for ordinary reinforcement-learning algorithms do not apply for the discretized state space. This holds for any function approximation that uses a feature space that is not an injective function of the Markov state space.

Intuitively, this point can be explained with a simple example. Consider a state space $S = \mathbb{R}$ that is discretized such that $\phi(s) = (1, 0, 0)^T$ when $s \leq -2$, $\phi(s) = (0, 1, 0)^T$ when $-2 < s < 2$ and $\phi(s) = (0, 0, 1)^T$ when $s \geq 2$. The action space is $A = \{-2, 2\}$, the transition function is $s_{t+1} = s_t + a_t$ and the initial state is $s_0 = 1$. The reward is defined by $r_{t+1} = 1$ if $s_t \in (-2, 2)$ and $r_{t+1} = -1$ otherwise. The reward function and the feature mapping are shown in Figure 7.2. In this MDP, it is optimal to jump back and forth between the states $s = -1$ and $s = 1$. However, if we observe the feature vector $(0, 1, 0)^T$, we can not know if we are in $s = -1$ or $s = 1$ and we cannot determine the optimal action.

Another practical issue with methods such as tile coding is related to the step-size parameter that many algorithms use. For instance, in many algorithms the parameters of a linear function approximator are updated with an update akin to

⁴ Note that $1 < M < D_\phi$ implies that $D_\phi < \binom{D_\phi}{M}$.

$$\theta_{t+1} = \theta_t + \alpha_t(s_t) \delta_t \phi(s_t) , \quad (7.4)$$

where $\alpha_t(s_t) \in [0,1]$ is a step size and δ_t is an error for the value of the current state. This may be a temporal-difference error, the difference between the current value and a Monte Carlo sample, or any other relevant error. A derivation and explanation of this update and variants thereof are given below, in Sections 7.2.3.1 and 7.3.1.2.

If we look at the update to a value $V(s) = \theta^T \phi(s)$ that results from (7.4), we get

$$\begin{aligned} V_{t+1}(s) &= \theta_{t+1}^T \phi(s) = (\theta_t + \alpha_t(s) \delta_t \phi(s))^T \phi(s) \\ &= \theta_t^T \phi(s) + \alpha_t(s) \phi^T(s) \phi(s) \delta_t \\ &= V_t(s) + \alpha_t(s) \phi^T(s) \phi(s) \delta_t . \end{aligned}$$

In other words, the effective step size for the values is equal to

$$\alpha_t(s_t) \phi^T(s_t) \phi(s_t) = \alpha_t(s_t) \|\phi(s_t)\| . \quad (7.5)$$

For instance, in tile coding $\|\phi(s_t)\|$ is equal to the number of tilings M . Therefore, the effective step size on the value function is larger than one for $\alpha_t(s_t) > 1/M$. This can cause divergence of the parameters. Conversely, if the euclidean norm $\|\phi(s)\|$ of the feature vector is often small, the change to the value function may be smaller than intended.

This issue can occur for any feature space and linear function approximation, since then the effective step sizes in (7.5) are used for the update to the value function. This indicates that it can be a good idea to scale the step size appropriately, by using

$$\tilde{\alpha}_t(s_t) = \alpha_t(s_t) / \|\phi(s_t)\| ,$$

where $\tilde{\alpha}_t(s_t)$ is the scaled step size.⁵ This scaled step size can prevent unintended small as well as unintended large updates to the values.

In general, it is often a good idea to make sure that $|\phi(s)| = \sum_k^{D_\Phi} \phi_k(s) \leq 1$ for all s . For instance, in tile coding we could set the value of active features equal to $1/M$ instead of to 1. In general, function approximators for which this property holds include so called *averagers* (Gordon, 1995) and *interpolators* (Szepesvári and Smart, 2004). Such feature representations have good convergence properties, because they are non-expansions, which means that $\max_s |\phi(s)^T \theta - \phi(s)^T \theta'| \leq \max_k |\theta_k - \theta'_k|$ for any feature vector $\phi(s)$ and any two parameter vectors θ and θ' . A non-expansive function makes it easier to prove that an algorithm iteratively improves its solution in expectation through a so-called contraction mapping (Gordon, 1995; Littman and Szepesvári, 1996; Bertsekas and Tsitsiklis, 1996; Bertsekas, 2007; Szepesvári, 2010; Buşoniu et al., 2010). Algorithms that implement a contraction mapping eventually reach an optimal solution and can be guaranteed not to diverge, for instance by updating their parameters to infinitely high values.

⁵ One can safely define $\tilde{\alpha}_t(s_t) = 0$ if $\|\phi(s_t)\| = 0$, since in that case update (7.4) would not change the parameters anyway.

A final issue with discretization is that it introduces discontinuities in the function. If the input changes a small amount, the approximated value may change a fairly large amount if the two inputs fall into different segments of the input space.

7.2.1.3 Fuzzy Representations

Some of the issues with discretization can be avoided by using a function that is piece-wise linear, rather than piece-wise constant. One way to do this, is by using so-called fuzzy sets (Zadeh, 1965; Klir and Yuan, 1995; Babuska, 1998). A fuzzy set is a generalization of normal sets to fuzzy membership. This means that elements can partially belong to a set, instead of just the possibilities of truth or falsehood.

A common example of fuzzy sets is the division of temperature into ‘cold’ and ‘warm’. There is a gradual transition between cold and warm, so often it is more natural to say that a certain temperature is partially cold and partially warm.

In reinforcement learning, the state or state-action space can be divided into fuzzy sets. Then, a state may belong partially to the set defined by feature ϕ_i and partially to the set defined by feature ϕ_j . For instance, we may have $\phi_i(s) = 0.1$ and $\phi_j(s) = 0.3$. An advantage of this view is that it is quite natural to assume that $\sum_k \phi_k(s) \leq 1$, since each part of an element can belong to only one set. For instance, something cannot be fully warm and fully cold at the same time.

It is possible to define the sets such that each combination of feature activations corresponds precisely to one single state, thereby avoiding the partial-observability problem sketched earlier. A common choice is to use triangular functions that are equal to one at the center of the corresponding feature and decay linearly to zero for states further from the center. With some care, such features can be constructed such that they span the whole state space and $\sum_k \phi_k(s) \leq 1$ for all states.

A full treatment of fuzzy reinforcement learning falls outside the scope of this chapter. References that make the explicit connection between fuzzy logic and reinforcement learning include Berenji and Khedkar (1992); Berenji (1994); Lin and Lee (1994); Glorenc (1994); Bonarini (1996); Jouffe (1998); Zhou and Meng (2003) and Buşoniu et al (2008, 2010). A drawback of fuzzy sets is that these sets still need to be defined beforehand, which may be difficult.

7.2.2 Non-linear Function Approximation

The main drawback of linear function approximation compared to non-linear function approximation is the need for good informative features.⁶ The features are often assumed to be hand-picked beforehand, which may require domain knowledge. Even if convergence in the limit to an optimal solution is guaranteed, this solution is only optimal in the sense that it is the best possible linear function of the given features. Additionally, while less theoretical guarantees can be given, nice empirical results have been obtained by combining reinforcement-learning algorithms with

⁶ Non-parametric approaches somewhat alleviate this point, but are harder to analyze in general. A discussion on such methods falls outside the scope of this chapter.

non-linear function approximators, such as neural networks (Haykin, 1994; Bishop, 1995, 2006; Ripley, 2008). Examples include Backgammon (Tesauro, 1992, 1994, 1995), robotics (Anderson, 1989; Lin, 1993; Touzet, 1997; Coulom, 2002) and elevator dispatching (Crites and Barto, 1996, 1998).

In a parametric non-linear function approximator, the function that should be optimized is represented by some predetermined parametrized function. For instance, for value-based algorithms we may have

$$V_t(s) = V(\phi(s), \theta_t) . \quad (7.6)$$

Here the size of $\theta_t \in \Theta$ is not necessarily equal to the size of $\phi(s) \in \Phi$. For instance, V may be a neural network where θ_t is a vector with all its weights at time t . Often, the functional form of V is fixed. However, it is also possible to change the structure of the function during learning (e.g., Stanley and Miikkulainen, 2002; Taylor et al, 2006; Whiteson and Stone, 2006; Buşoniu et al, 2010).

In general, a non-linear function approximator may approximate an unknown function with better accuracy than a linear function approximator that uses the same input features. In some cases, it is even possible to avoid defining features altogether by using the state variables as inputs. A drawback of non-linear function approximation in reinforcement learning is that less convergence guarantees can be given. In some cases, convergence to a local optimum can be assured (e.g., Maei et al, 2009), but in general the theory is less well developed than for linear approximation.

7.2.3 *Updating Parameters*

Some algorithms allow for the closed-form computation of parameters that best approximate the desired function, for a given set of experience samples. For instance, when TD-learning is coupled with linear function approximation, least-squares temporal-difference learning (LSTD) (Bradtko and Barto, 1996; Boyan, 2002; Geramifard et al, 2006) can be used to compute parameters that minimize the empirical temporal-difference error over the observed transitions. However, for non-linear algorithms such as Q-learning or when non-linear function approximation is used, these methods are not applicable and the parameters should be optimized in a different manner.

Below, we explain how to use the two general techniques of gradient descent and gradient-free optimization to adapt the parameters of the approximations. These procedures can be used with both linear and non-linear approximation and they can be used for all three types of functions: models, value functions and policies. In Section 7.3, we discuss reinforcement-learning algorithms that use these methods.

We will not discuss Bayesian methods in any detail, but such methods can be used to learn the probability distributions of stationary functions, such as the reward and transition functions of a stationary MDP. An advantage of this is that the exploration of an online algorithm can choose actions to increase the knowledge of parts of the model that have high uncertainty. Bayesian methods are somewhat less suited to

```

1: input: differentiable function  $\mathbf{E} : \mathbb{R}^N \times \mathbb{R}^P \rightarrow \mathbb{R}$  to be minimized,  

   step size sequence  $\alpha_t \in [0,1]$ , initial parameters  $\theta_0 \in \mathbb{R}^P$   

2: output: a parameter vector  $\theta$  such that  $\mathbf{E}$  is small  

3: for all  $t \in \{1,2,\dots\}$  do  

4:   Observe  $x_t$ ,  $\mathbf{E}(x_t, \theta_t)$   

5:   Calculate gradient:  


```

$$\nabla_{\theta} \mathbf{E}(x_t, \theta_t) = \left(\frac{\partial}{\partial \theta_t[1]} \mathbf{E}(x_t, \theta_t), \dots, \frac{\partial}{\partial \theta_t[P]} \mathbf{E}(x_t, \theta_t) \right)^T .$$

```

6:   Update parameters:  


$$\theta_{t+1} = \theta_t - \alpha_t \nabla_{\theta} \mathbf{E}(x_t, \theta_t)$$


```

Algorithm 15. Stochastic gradient descent

learn the value of non-stationary functions, such as the value of a changing policy. For more general information about Bayesian inference, see for instance Bishop (2006). For Bayesian methods in the context of reinforcement learning, see Dearden et al (1998, 1999); Strens (2000); Poupart et al (2006) and Chapter 11.

7.2.3.1 Gradient Descent

A gradient-descent update follows the direction of the negative gradient of some parametrized function that we want to minimize. The gradient of a parametrized function is a vector in parameter space that points in the direction in which the function increases, according to a first-order Taylor expansion. Put more simply, if the function is smooth and we change the parameters a small amount in the direction of the gradient, we expect the function to increase slightly.

The negative gradient points in the direction in which the function is expected to decrease, so moving the parameters in this direction should result in a lower value for the function. Algorithm 15 shows the basic algorithm, where for simplicity a real-valued parametrized function $\mathbf{E} : \mathbb{R}^N \times \mathbb{R}^P \rightarrow \mathbb{R}$ is considered. The goal is to make the output of this function small. To do this, the parameters of $\theta \in \mathbb{R}^{D_{\theta}}$ of \mathbf{E} are updated in the direction of the negative gradient. The gradient $\nabla_{\theta} \mathbf{E}(x, \theta)$ is a column vector whose components are the derivatives of \mathbf{E} to the elements of the parameter vector θ , calculated at the input x . Because the gradient only describes the local shape of the function, this algorithm can end up in a local minimum.

Usually, \mathbf{E} is an error measure such as a temporal-difference or a prediction error. For instance, consider a parametrized approximate reward function $\bar{R} : S \times A \times \mathbb{R}^P \rightarrow \mathbb{R}$ and a sample (s_t, a_t, r_{t+1}) . Then, we might use $\mathbf{E}(s_t, a_t, \theta_t) = (\bar{R}(s_t, a_t, \theta_t) - r_{t+1})^2$.

If the gradient is calculated over more than one input-output pair at the same time, the result is the following batch update

$$\theta_{t+1} = \theta_t - \alpha_t \sum_i \nabla_{\theta} \mathbf{E}_i(x_i, \theta_t) ,$$

where $\mathbf{E}_i(x_i, \theta_t)$ is the error for the i^{th} input x_i and $\alpha_t \in [0, 1]$ is a step-size parameter. If the error is defined over only a single input-output pair, the update is called a stochastic gradient descent update. Batch updates can be used in offline algorithms, while stochastic gradient descent updates are more suitable for online algorithms.

There is some indication that often stochastic gradient descent converges faster than batch gradient descent (Wilson and Martinez, 2003). Another advantage of stochastic gradient descent over batch learning is that it is straightforward to extend online stochastic gradient descent to non-stationary targets, for instance if the policy changes after an update. These features make online gradient methods quite suitable for online reinforcement learning. In general, in combination with reinforcement learning convergence to an optimal solution is not guaranteed, although in some cases convergence to a local optimum can be proven (Maei et al, 2009).

In the context of neural networks, gradient descent is often implemented through backpropagation (Bryson and Ho, 1969; Werbos, 1974; Rumelhart et al, 1986), which uses the chain rule and the layer structure of the networks to efficiently calculate the derivatives of the network's output to its parameters. However, the principle of gradient descent can be applied to any differentiable function.

In some cases, the normal gradient is not the best choice. More formally, a problem of ordinary gradient descent is that the distance metric in parameter space may differ from the distance metric in function space, because of interactions between the parameters. Let $d\theta \in \mathbb{R}^P$ denote a vector in parameter space. The Euclidean norm of this vector is $\|d\theta\| = d\theta^T d\theta$. However, if the parameter space is a curved space—known as a Riemannian manifold—it is more appropriate to use $d\theta^T G d\theta$ where G is a $P \times P$ positive semi-definite matrix. With this weighted distance metric, the direction of steepest descent becomes

$$\tilde{\nabla}_{\theta} \mathbf{E}(x, \theta) = G^{-1} \nabla_{\theta} \mathbf{E}(x, \theta) ,$$

which is known as the *natural gradient* (Amari, 1998). In general, the best choice for matrix G depends on the functional form of \mathbf{E} . Since \mathbf{E} is not known in general, G will usually need to be estimated.

Natural gradients have a number of advantages. For instance, the natural gradient is invariant to transformations of the parameters. In other words, when using a natural gradient the change in our function does not depend on the precise parametrization of the function. This is somewhat similar to our observation in Section 7.2.1.2 that we can scale the step size to tune the size of the step in value space rather than in parameter space. Only here we consider the direction of the update to the parameters, rather than its size. Additionally, the natural gradient avoids plateaus in function space, often resulting in faster convergence. We discuss natural gradients in more detail when we discuss policy-gradient algorithms in Section 7.3.2.1.

7.2.3.2 Gradient-Free Optimization

Gradient-free methods are useful when the function that is optimized is not differentiable or when it is expected that many local optima exist. Many general global

methods for optimization exist, including evolutionary algorithms (Holland, 1962; Rechenberg, 1971; Holland, 1975; Schwefel, 1977; Davis, 1991; Bäck and Schwefel, 1993), simulated annealing (Kirkpatrick, 1984), particle swarm optimization (Kennedy and Eberhart, 1995) and cross-entropy optimization (Rubinstein, 1999; Rubinstein and Kroese, 2004). Most of these methods share some common features that we will outline below. We focus on cross-entropy and a subset of evolutionary algorithms, but the other approaches can be used quite similarly. For introductions to evolutionary algorithms, see the books by Bäck (1996) and Eiben and Smith (2003). For a more extensive account on evolutionary algorithms in reinforcement learning, see Chapter 10. We give a short overview of how such algorithms work.

All the methods described here use a population of solutions. Traditional evolutionary algorithms create a population of solutions and adapt this population by selecting some solutions, recombining these and possibly mutating the result. The newly obtained solutions then replace some or all of the solutions in the old population. The selection procedure typically takes into account the fitness of the solutions, such that solutions with higher quality have a larger probability of being used to create new solutions.

Recently, it has become more common to adapt the parameters of a probability distribution that generates solutions, rather than to adapt the solutions themselves. This approach is used in so-called evolutionary strategies (Bäck, 1996; Beyer and Schwefel, 2002). Such approaches generate a population, but use the fitness of the solutions to adapt the parameters of the generating distribution, rather than the solutions themselves. A new population is then obtained by generating new solutions from the adapted probability distribution. Some specific algorithms include the following. *Covariance matrix adaptation evolution strategies* (CMA-ES) (Hansen and Ostermeier, 2001) weigh the sampled solutions according to their fitness and use the weighted mean as the mean of the new distribution. *Natural evolutionary strategies* (NES) (Wierstra et al, 2008; Sun et al, 2009) use all the generated solutions to estimate the gradient of the parameters of the generating function, and then use natural gradient ascent to improve these parameters. *Cross-entropy optimization* methods (Rubinstein and Kroese, 2004) simply select the m solutions with the highest fitness—where m is a parameter—and use the mean of these solutions to find a new mean for the distribution.⁷

⁷ According to this description, cross-entropy optimization can be considered an evolutionary strategy similar to CMA-ES, using a special weighting that weighs the top m solutions with $1/m$ and the rest with zero. There are more differences between the known algorithmic implementations however, most important of which is perhaps the more elegant estimation of the covariance matrix of the newly formed distribution by CMA-ES, aimed to increase the probability of finding new solutions with high fitness. Some versions of cross-entropy add noise to the variance to prevent premature convergence (e.g., Szita and Lörincz, 2006), but the theory behind this seems less well-developed than covariance estimation used by CMA-ES.

On a similar note, it has recently been shown that CMA-ES and NES are equivalent except for some differences in the proposed implementation of the algorithms (Akimoto et al, 2011).

```

1: input: parametrized population PDF  $p : \mathbb{R}^K \times \mathbb{R}^P \rightarrow \mathbb{R}$ , fitness function  $f : \mathbb{R}^P \rightarrow \mathbb{R}$ ,  

   initial parameters  $\zeta_0 \in \mathbb{R}^K$ , population size  $n$   

2: output: a parameter vector  $\zeta$  such that if  $\theta \sim p(\zeta, \cdot)$  then  $f(\theta)$  is large with high  

   probability  

3: for all  $t \in \{1, 2, \dots\}$  do  

4:   Construct population  $\Theta_t = \{\bar{\theta}_1, \bar{\theta}_2, \dots, \bar{\theta}_n\}$ , where  $\bar{\theta}_i \sim p(\zeta_t, \cdot)$   

5:   Use the fitness scores  $f(\bar{\theta}_i)$  to compute  $\zeta_{t+1}$  such that  $E\{f(\theta) | \zeta_{t+1}\} > E\{f(\theta) | \zeta_t\}$ 

```

Algorithm 16. A generic evolutionary strategy

A generic evolutionary strategy is shown in Algorithm 16. The method to compute the next parameter setting ζ_{t+1} for the generating function in line 5 differs between algorithms. However, all attempt to increase the expected fitness such that $E\{f(\theta) | \zeta_{t+1}\}$ is higher than the expected fitness of the former population $E\{f(\theta) | \zeta_t\}$. These expectations are defined by

$$E\{f(\theta) | \zeta\} = \int_{\mathbb{R}^P} p(\zeta, \theta) f(\theta) d\theta .$$

Care should be taken that the variance of the distribution does not become too small too quickly, in order to prevent premature convergence to sub-optimal solutions. A simple way to do this, is by using a step-size parameter (Rubinstein and Kroese, 2004) on the parameters in order to prevent from too large changes per iteration. More sophisticated methods to prevent premature convergence include the use of the natural gradient by NES, and the use of enforced correlations between the covariance matrices of consecutive populations by CMA-ES.

No general guarantees can be given concerning convergence to the optimal solution for evolutionary strategies. Convergence to the optimal solution for non-stationary problems, such as the control problem in reinforcement learning, seems even harder to prove. Despite this lack of guarantees, these methods can perform well in practice. The major bottleneck is usually that the computation of the fitness can be both noisy and expensive. Additionally, these methods have been designed mostly with stationary optimization problems in mind. Therefore, they are more suited to optimize a policy using Monte Carlo samples than to approximate the value of the unknown optimal policy. In Section 7.4, we compare the performance of CMA-ES and an actor-critic temporal-difference approach.

The gradient-free methods mentioned above all fall into a category known as *metaheuristics* (Glover and Kochenberger, 2003). These methods iteratively search for good candidate solutions, or a distribution that generates these. Another approach is to construct an easier solvable (e.g., quadratic) model of the function that is to be optimized and then maximize this model analytically (see, e.g., Powell, 2002, 2006; Huyer and Neumaier, 2008). New samples can be iteratively chosen to improve the approximate model. We do not know any papers that have used such methods in a reinforcement learning context, but the sample-efficiency of such

methods in high-dimensional problems make them an interesting direction for future research.

7.3 Approximate Reinforcement Learning

In this section we apply the general function approximation techniques described in Section 7.2 to reinforcement learning. We discuss some of the current state of the art in reinforcement learning in continuous domains. As mentioned earlier in this chapter, we will not discuss the construction of approximate models because even if a model is known exact planning is often infeasible in continuous spaces.

7.3.1 Value Approximation

In value-approximation algorithms, experience samples are used to update a value function that gives an approximation of the current or the optimal policy. Many reinforcement-learning algorithms fall into this category. Important differences between algorithms within this category is whether they are on-policy or off-policy and whether they update online or offline. Finally, a value-approximation algorithm may store a state-value function $V : S \rightarrow \mathbb{R}$, or an action-value function $Q : S \times A \rightarrow \mathbb{R}$, or even both (Wiering and van Hasselt, 2009). We will explain these properties and give examples of algorithms for each combination of properties.

On-policy algorithms approximate the state-value function V^π or the action-value function Q^π , which represent the value of the policy π that they are currently following. Although the optimal policy π^* is unknown initially, such algorithms can eventually approximate the optimal value function V^* or Q^* by using policy iteration, which improves the policy between evaluation steps. Such policy improvements may occur as often as each time step. *Off-policy* algorithms can learn about the value of a different policy than the one that is being followed. This is useful, as it means we do not have to follow a (near-) optimal policy to learn about the value of the optimal policy.

Online algorithms adapt their value approximation after each observed sample. *Offline* algorithms operate on batches of samples. Usually, online algorithms require much less computation per sample, whereas offline algorithms require less samples to reach a similar accuracy of the approximation.

Online on-policy algorithms include temporal-difference (TD) algorithms, such as TD-learning (Sutton, 1984, 1988), Sarsa (Rummery and Niranjan, 1994; Sutton and Barto, 1998) and Expected-Sarsa (van Seijen et al, 2009).

Offline on-policy algorithms include least-squares approaches, such as least-squared temporal difference (LSTD) (Bradtko and Barto, 1996; Boyan, 2002; Geramifard et al, 2006), least-squares policy evaluation (LSPE) (Nedić and Bertsekas, 2003) and least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003). Because of limited space we will not discuss least-squares approaches in this chapter, but see Chapter 3 of this volume.

Arguably the best known model-free online off-policy algorithm is *Q-learning* (Watkins, 1989; Watkins and Dayan, 1992). Its many derivatives include Perseus (Spaan and Vlassis, 2005), Delayed Q-learning (Strehl et al, 2006) and Bayesian Q-learning (Dearden et al, 1998; see also Chapter 11). All these variants try to estimate the optimal policy through use of some variant of the Bellman optimality equation. In general, off-policy algorithms need not estimate the optimal policy, but can also approximate an arbitrary other policy (Precup et al, 2000; Precup and Sutton, 2001; Sutton et al, 2008; van Hasselt, 2011, Section 5.4). Offline variants of Q-learning include fitted Q-iteration (Ernst et al, 2005; Riedmiller, 2005; Antos et al, 2008a).

An issue with both the online and the offline variants of Q-learning is that noise in the value approximations, due to the stochasticity of the problem and the limitations of the function approximator, can result a structural overestimation bias. In short, the value of $\max_a Q_t(s,a)$, as used by Q-learning, may—even in expectancy—be far larger than $\max_a Q^*(s,a)$. This bias can severely slow convergence of Q-learning, even in tabular settings (van Hasselt, 2010) and if care is not taken with the choice of function approximator, it may result in divergence of the parameters (Thrun and Schwartz, 1993). A partial solution for this bias is given by the Double Q-learning algorithm (van Hasselt, 2010), where two action-value functions produce an estimate which may underestimate $\max_a Q^*(s,a)$, but is bounded in expectancy.

Many of the aforementioned algorithms can be used both online and offline, but are better suited for either of these approaches. For instance, fitted Q-iteration usually is used as an offline algorithm, since the algorithm is considered too computationally expensive to be run after each sample. Conversely, online algorithms can store the observed samples and reuse these as if they were observed again in a form of experience replay (Lin, 1992). The least-squares and fitted variants are usually used as offline versions of temporal-difference algorithms. There are exceptions however, such as the online incremental LSTD algorithm (Geramifard et al, 2006, 2007).

If the initial policy does not easily reach some interesting parts of the state-space, online algorithms have the advantage that the policy is usually updated more quickly, because value updates are not delayed until a sufficiently large batch of samples is obtained. This means that online algorithms are sometimes more sample-efficient in control problems.

In the next two subsections, we discuss in detail some online value-approximation algorithms that use a gradient-descent update on a predefined error measure.

7.3.1.1 Objective Functions

In order to update a value with gradient descent, we must choose some measure of error that we can minimize. This measure is often referred to as the *objective function*. To be able to reason more formally about these objective functions, we introduce the concepts of *function space* and *projections*. Recall that \mathcal{V} is the space of value functions, such that $V \in \mathcal{V}$. Let $\mathcal{F} \subseteq \mathcal{V}$ denote the function space of representable functions for some function approximator. Intuitively, if \mathcal{F} contains

a large subset of \mathcal{V} , the function is flexible and can accurately approximate many value functions. However, it may be prone to overfitting of the perceived data and it may be slow to update since usually a more flexible function requires more tunable parameters. Conversely, if \mathcal{F} is small compared to \mathcal{V} , the function is not very flexible. For instance, the function space of a linear approximator is usually smaller than that of a non-linear approximator. A parametrized function has a parameter vector $\theta = \{\theta[1], \dots, \theta[D_\theta]\} \in \mathbb{R}^{D_\theta}$ that can be adjusted during training. The function space is then defined by

$$\mathcal{F} = \{V(\cdot, \theta) | \theta \in \mathbb{R}^{D_\theta}\} .$$

From here on further, we denote parametrized value functions by V_t if we want to stress the dependence on time and by V^θ if we want to stress the dependence on the parameters. By definition, $V_t(s) = V(s, \theta_t)$ and $V^\theta(s) = V(s, \theta)$.

A projection $\Pi : \mathcal{V} \rightarrow \mathcal{F}$ is an operator that maps a value function to the closest representable function in \mathcal{F} , under a certain norm. This projection is defined by

$$\|V - \Pi V\|_w = \min_{v \in \mathcal{F}} \|V - v\|_w = \min_{\theta} \|V - V^\theta\|_w ,$$

where $\|\cdot\|_w$ is a weighted norm. We assume the norm is quadratic, such that

$$\|V - V^\theta\|_w = \int_{s \in S} w(s) (V(s) - V^\theta(s))^2 ds .$$

This means that the projection is determined by the functional form of the approximator and the weights of the norm.

Let $B = B^\pi$ or $B = B^*$, depending on whether we are approximating the value of a given policy, or the value of the optimal policy. It is often not possible to find a parameter vector that fulfills the Bellman equations $V^\theta = BV^\theta$ for the whole state space exactly, because the value BV^θ may not be representable with the chosen function. Rather, the best we can hope for is a parameter vector that fulfills

$$V^\theta = \Pi BV^\theta . \quad (7.7)$$

This is called the projected Bellman equation; Π projects the outcome of the Bellman operator back to the space that is representable by the function approximation.

In some cases, it is possible to give a closed form expression for the projection (Tsitsiklis and Van Roy, 1997; Bertsekas, 2007; Szepesvári, 2010). For instance, consider a finite state space with N states and a linear function $V_t(s) = \theta^T \phi(s)$, where $D_\theta = D_\phi \ll N$. Let $p_s = P(s_t = s)$ denote the expected steady-state probabilities of sampling each state and store these values in a diagonal $N \times N$ matrix P . We assume the states are always sampled according to these fixed probabilities. Finally, the $N \times D_\phi$ matrix Φ holds the feature vectors for all states in its rows, such that $V_t = \Phi \theta_t$ and $V_t(s) = \Phi_s \theta_t = \theta_t^T \phi(s)$. Then, the projection operator can be represented by the $N \times N$ matrix

$$\Pi = \Phi (\Phi^T P \Phi)^{-1} \Phi^T P . \quad (7.8)$$

The inverse exists if the features are linearly independent, such that Φ has rank D_Φ .

With this definition $\Pi V_t = \Pi \Phi \theta_t = \Phi \theta_t = V_t$, but $\Pi B V_t \neq B V_t$, unless $B V_t$ can be expressed as a linear function of the feature vectors. A projection matrix as defined in (7.8) is used in the analysis and in the derivation of several algorithms (Tsitsiklis and Van Roy, 1997; Nedić and Bertsekas, 2003; Bertsekas et al, 2004; Sutton et al, 2008, 2009; Maei and Sutton, 2010). We discuss some of these in the next section.

7.3.1.2 Gradient Temporal-Difference Learning

We generalize standard temporal-difference learning (*TD-learning*) (Sutton, 1984, 1988) to a gradient update on the parameters of a function approximator. The tabular TD-learning update is

$$V_{t+1}(s_t) = V_t(s_t) + \alpha_t(s_t) \delta_t ,$$

where $\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$ and $\alpha_t(s) \in [0,1]$ is a step-size parameter. When TD-learning is used to estimate the value of a given stationary policy under on-policy updates the value function converges when the feature vectors are linearly independent (Sutton, 1984, 1988). Later it was shown that TD-learning also converges when eligibility traces are used and when the features are not linearly independent (Dayan, 1992; Peng, 1993; Dayan and Sejnowski, 1994; Bertsekas and Tsitsiklis, 1996; Tsitsiklis and Van Roy, 1997). More recently, variants of TD-learning were proposed that converge under off-policy updates (Sutton et al, 2008, 2009; Maei and Sutton, 2010). We discuss these variants below. A limitation of most aforementioned results is that they apply only to the prediction setting. Recently some work has been done to extend the analysis to the control setting. This has led to the Greedy-GQ algorithm, which extends Q-learning to linear function approximation without the danger of divergence, under some conditions (Maei et al, 2010).

When the state values are stored in a table, TD-learning can be interpreted as a stochastic gradient-descent update on the one-step temporal-difference error

$$\mathbf{E}(s_t) = \frac{1}{2} (r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t))^2 = \frac{1}{2} (\delta_t)^2 . \quad (7.9)$$

If V_t is a parametrized function such that $V_t(s) = V(s, \theta_t)$, the negative gradient with respect to the parameters is given by

$$-\nabla_\theta \mathbf{E}(s_t, \theta) = -(r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)) \nabla_\theta (r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)) .$$

Apart from the state and the parameters, the error depends on the MDP and the policy. We do not specify these dependencies explicitly to avoid cluttering the notation.

A direct implementation of gradient descent based on the error in (7.9) would adapt the parameters to move $V_t(s)$ closer to $r_{t+1} + \gamma V_t(s_{t+1})$ as desired, but would also move $\gamma V_t(s_{t+1})$ closer to $V_t(s_t) - r_{t+1}$. Such an algorithm is called a residual-gradient algorithm (Baird, 1995). Alternatively, we can interpret $r_{t+1} + \gamma V_t(s_{t+1})$ as

a stochastic approximation for V^π that does not depend on θ . Then, the negative gradient is (Sutton, 1984, 1988)

$$-\nabla_\theta \mathbf{E}_t(s_t, \theta) = (r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)) \nabla_\theta V_t(s_t) .$$

This implies the parameters can be updated as

$$\theta_{t+1} = \theta_t + \alpha_t(s_t) \delta_t \nabla_\theta V_t(s_t) . \quad (7.10)$$

This is the conventional TD learning update and it usually converges faster than the residual-gradient update (Gordon, 1995, 1999). For linear function approximation, for any θ we have $\nabla_\theta V_t(s_t) = \phi(s_t)$ and we obtain the same update as was shown earlier for tile coding in (7.4). Similar updates for action-value algorithms are obtained by replacing $\nabla_\theta V_t(s_t)$ in (7.10) with $\nabla_\theta Q_t(s_t, a_t)$ and using, for instance

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) , \text{ or} \\ \delta_t &= r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) , \end{aligned}$$

for Q-learning and Sarsa, respectively.

We can incorporate accumulating eligibility traces with trace parameter λ with the following two equations (Sutton, 1984, 1988):

$$\begin{aligned} \mathbf{e}_{t+1} &= \lambda \gamma \mathbf{e}_t + \nabla_\theta V_t(s_t) , \\ \theta_{t+1} &= \theta_t + \alpha_t(s_t) \delta_t \mathbf{e}_{t+1} , \end{aligned}$$

where $\mathbf{e} \in \mathbb{R}^{D_\Phi}$ is a trace vector. Replacing traces (Singh and Sutton, 1996) are less straightforward, although the suggestion by Främling (2007) seem sensible:

$$\mathbf{e}_{t+1} = \max(\lambda \gamma \mathbf{e}_t, \nabla_\theta V_t(s_t)) ,$$

since this corresponds nicely to the common practice for tile coding and this update reduces to the conventional replacing traces update when the values are stored in a table. However, a good theoretical justification for this update is still lacking.

Parameters updated with (7.10) may diverge when off-policy updates are used. This holds for any temporal-difference method with $\lambda < 1$ when we use linear (Baird, 1995) or non-linear function approximation (Tsitsiklis and Van Roy, 1996). In other words, if we sample transitions from a distribution that does not comply completely to the state-visit probabilities that would occur under the estimation policy, the parameters of the function may diverge. This is unfortunate, because in the control setting ultimately we want to learn about the unknown optimal policy.

Recently, a class of algorithms has been proposed to deal with this issue (Sutton et al, 2008, 2009; Maei et al, 2009; Maei and Sutton, 2010). The idea is to perform a stochastic gradient-descent update on the quadratic projected temporal difference:

$$\mathbf{E}(\theta) = \frac{1}{2} \|V_t - \Pi B V_t\|_P = \frac{1}{2} \int_{s \in S} P(s=s_t) (V_t(s) - \Pi B V_t(s))^2 ds . \quad (7.11)$$

In contrast with (7.9), this error does not depend on the time step or the state. The norm in (7.11) is weighted according to the state probabilities that are stored in the diagonal matrix P , as described in Section 7.3.1.1. If we minimize (7.11), we reach the fixed point in (7.7). To do this, we rewrite the error to

$$\mathbf{E}(\theta_t) = \frac{1}{2} (E\{\delta_t \nabla_\theta V_t(s)\})^T (E\{\nabla_\theta V_t(s) \nabla_\theta^T V_t(s)\})^{-1} E\{\delta_t \nabla_\theta V_t(s)\} , \quad (7.12)$$

where it is assumed that the inverse exists (Maei et al, 2009). The expectancies are taken over the state probabilities in P . The error is the product of multiple expected values. These expected values can not be sampled from a single experience, because then the samples would be correlated. This can be solved by updating an additional parameter vector. We use the shorthands $\phi = \phi(s_t)$ and $\phi' = \phi(s_{t+1})$ and we assume linear function approximation. Then $\nabla_\theta V_t(s_t) = \phi$ and we get

$$\begin{aligned} -\nabla_\theta \mathbf{E}(\theta_t) &= E\{(\phi - \gamma\phi')\phi^T\} (E\{\phi\phi^T\})^{-1} E\{\delta_t\phi\} \\ &\approx E\{(\phi - \gamma\phi')\phi^T\} \mathbf{w} , \end{aligned}$$

where $\mathbf{w}_t \in \mathbb{R}^{D_\phi}$ is an additional parameter vector. This vector should approximate $(E\{\phi\phi^T\})^{-1} E\{\delta_t\phi\}$, which can be done with the update

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \beta_t(s_t) (\delta_t - \phi^T \mathbf{w}_t) \phi ,$$

where $\beta_t(s_t) \in [0,1]$ is a step-size parameter. Then there is only one expected value left to approximate, which can be done with a single sample. This leads to the update

$$\theta_{t+1} = \theta_t + \alpha_t(s_t) (\phi - \gamma\phi') (\phi^T \mathbf{w}_t) ,$$

which is called the GTD2 (Gradient Temporal-Difference Learning, version 2) algorithm (Sutton et al, 2009). One can also write the gradient in a slightly different manner to obtain the similar TDC algorithm, which is defined as:

$$\theta_{t+1} = \theta_t + \alpha_t(s_t) (\delta_t\phi - \gamma\phi' (\phi^T \mathbf{w}_t)) ,$$

where \mathbf{w}_t is updated as above. This algorithm is named TD with gradient correction (TDC), because the update to the primary parameter vector θ_t is equal to (7.10), except for a correction term. This term prevents divergence of the parameters when off-policy updates are used. Both GTD2 and TDC can be shown to minimize (7.12), if the states are sampled according to P . The difference with ordinary TD-learning is that these algorithms also converge when the probabilities in P differ from those that result from following the policy π , whose value we are estimating. This is useful for instance when we have access to a simulator that allows us to sample the states in any order, while π would spend much time in uninteresting states.

When non-linear smooth function approximators are used, it can be proven that similar algorithms reach local optima (Maei et al, 2009). The updates for the non-linear algorithms are similar to the ones above, with another correction term. The

updates can be extended to a form of Q-learning in order to learn action values with eligibility traces. The resulting GQ(λ) algorithm is off-policy and converges to the value of a given estimation policy, even when the algorithm follows a different behavior policy (Maei and Sutton, 2010). The methods can be extended to control problems (Maei et al, 2010) with a greedy non-stationary estimation policy, although it is not yet clear how well the resulting Greedy-GQ algorithm performs in practice.

Although these theoretic insights and the resulting algorithms are promising, in practice the TD update in (7.10) is still the better choice in on-policy settings. Additionally, an update akin to (7.10) for Q-learning often results in good policies, although convergence can not be guaranteed in general. Furthermore, for specific functions—such as the earlier mentioned averagers—Q-learning does converge (Szepesvári and Smart, 2004). In practice, many problems do not have the precise characteristics that result in divergence of the parameters. Finally, the convergence guarantees are mainly limited to the use of samples from fixed steady-state probabilities.

If we can minimize the so-called Bellman residual error $\mathbf{E}(\theta_t) = \|V - BV\|_P$, this automatically minimizes the projected temporal-difference error in (7.11). Using $(\delta_t)^2$ as a sample for this error (with $B = B^\pi$) leads to a biased estimate, but other approaches have been proposed that use this error (Antos et al, 2008b; Maillard et al, 2010). It is problem-dependent whether minimizing the residual error leads to better results than minimizing the projected error (Scherrer, 2010).

It is non-trivial to extend the standard online temporal-difference algorithms such as Q-learning to continuous action spaces. Although we can construct an estimate of the value for each continuous action, it is non-trivial to find the maximizing action quickly when there are infinitely many actions. One way to do this is to simply discretize the action space, as in tile coding or by performing a line search (Pazis and Lagoudakis, 2009). Another method is to use interpolators, such as in wire-fitting (Baird and Klopff, 1993; Gaskett et al, 1999), which outputs a fixed number of candidate action-value pairs in each state. The actions and values are interpolated to form an estimate of the continuous action-value function in the current state. Because of the interpolation, the maximal value of the resulting function will always lie precisely on one of the candidate actions, thus facilitating the selection of the greedy action in the continuous space. However, the algorithms in the next section are usually much better suited for use in problems with continuous actions.

7.3.2 Policy Approximation

As discussed, determining a good policy from a model analytically can be intractable. An approximate state-action value function Q makes this easier, since then the greedy policy in each state s can be found by choosing the argument a that maximizes $Q(s,a)$. However, if the action space is continuous finding the greedy action in each state can be non-trivial and time-consuming. Therefore, it can be beneficial to store an explicit estimation of the optimal policy. In this section,

we consider actor-only and actor-critic algorithms that store a parametrized policy $\pi : S \times A \times \Psi \rightarrow [0,1]$, where $\pi(s, a, \psi)$ denotes the probability of selecting a in s for a given policy parameter vector $\psi \in \Psi \subseteq \mathbb{R}^{D_\psi}$. This policy is called an *actor*.

In Section 7.3.2.1 we discuss the general framework of policy-gradient algorithms and how this framework can be used to improve a policy. Then, in Section 7.3.2.3 we discuss actor-critic methods that use this framework along with an approximation of a value function. In Section 7.3.2.2 we discuss the application of evolutionary strategies for direct policy search. Finally, in Section 7.3.2.4 we discuss an alternative actor-critic method that uses a different type of update for its actor.

7.3.2.1 Policy-Gradient Algorithms

The idea of policy-gradient algorithms is to update the policy with gradient ascent on the cumulative expected value V^π (Williams, 1992; Sutton et al, 2000; Baxter and Bartlett, 2001; Peters and Schaal, 2008b; Rückstieß et al, 2010). If the gradient is known, we can update the policy parameters with

$$\psi_{k+1} = \psi_k + \beta_k \nabla_\psi E\{V^\pi(s_t)\} = \psi_k + \beta_k \nabla_\psi \int_{s \in S} P(s_t = s) V^\pi(s) ds .$$

Here $P(s_t = s)$ denotes the probability that the agent is in state s at time step t and $\beta_k \in [0,1]$ is a step size. In this update we use a subscript k in addition to t to distinguish between the time step of the actions and the update schedule of the policy parameters, which may not overlap. If the state space is finite, we can replace the integral with a sum.

As a practical alternative, we can use stochastic gradient descent:

$$\psi_{t+1} = \psi_t + \beta_t(s_t) \nabla_\psi V^\pi(s_t) . \quad (7.13)$$

Here the time step of the update corresponds to the time step of the action and we use the subscript t . Such procedures can at best hope to find a local optimum, because they use a gradient of a value function that is usually not convex with respect to the policy parameters. However, some promising results have been obtained, for instance in robotics (Benbrahim and Franklin, 1997; Peters et al, 2003).

The obvious problem with update (7.13) is that in general V^π is not known and therefore neither is its gradient. For a successful policy-gradient algorithm, we need an estimate of $\nabla_\psi V^\pi$. We will now discuss how to obtain such an estimate.

We will use the concept of a trajectory. A trajectory \mathcal{S} is a sequence of states and actions:

$$\mathcal{S} = \{s_0, a_0, s_1, a_1, \dots\} .$$

The probability that a given trajectory occurs is equal to the probability that the corresponding sequence of states and actions occurs with the given policy:

$$\begin{aligned} P(\mathcal{S}|s, \psi) &= P(s_0 = s)P(a_0|s_0, \psi)P(s_1|s_0, a_0)P(a_1|s_1, \psi)P(s_2|s_1, a_1)\dots \\ &= P(s_0 = s)\prod_{t=0}^{\infty}\pi(s_t, a_t, \psi)P_{s_t a_t}^{s_{t+1}}. \end{aligned} \quad (7.14)$$

The expected value V^π can then be expressed as an integral over all possible trajectories for the given policy and the corresponding expected rewards:

$$V^\pi(s) = \int_{\mathcal{S}} P(\mathcal{S}|s, \psi)E\left\{\sum_{t=0}^{\infty}\gamma^t r_{t+1} \middle| \mathcal{S}\right\} d\mathcal{S}.$$

Then, the gradient thereof can be expressed in closed form:

$$\begin{aligned} \nabla_\psi V^\pi(s) &= \int_{\mathcal{S}} \nabla_\psi P(\mathcal{S}|s, \psi)E\left\{\sum_{t=0}^{\infty}\gamma^t r_{t+1} \middle| \mathcal{S}\right\} d\mathcal{S} \\ &= \int_{\mathcal{S}} P(\mathcal{S}|s, \psi)\nabla_\psi \log P(\mathcal{S}|s, \psi)E\left\{\sum_{t=0}^{\infty}\gamma^t r_{t+1} \middle| \mathcal{S}\right\} d\mathcal{S} \\ &= E\left\{\nabla_\psi \log P(\mathcal{S}|s, \psi)E\left\{\sum_{t=0}^{\infty}\gamma^t r_{t+1} \middle| \mathcal{S}\right\} \middle| s, \psi\right\}, \end{aligned} \quad (7.15)$$

where we used the general identity $\nabla_x f(x) = f(x)\nabla_x \log f(x)$. This useful observation is related to Fisher's score function (Fisher, 1925; Rao and Poti, 1946) and the likelihood ratio (Fisher, 1922; Neyman and Pearson, 1928). It was applied to reinforcement learning by Williams (1992) for which reason it is sometimes called the REINFORCE trick, after the policy-gradient algorithm that was proposed therein (see, for instance, Peters and Schaal, 2008b).

The product in the definition of the probability of the trajectory as given in (7.14) implies that the logarithm in (7.15) consists of a sum of terms, in which only the policy terms depend on ψ . Therefore, the other terms disappear when we take the gradient and we obtain:

$$\begin{aligned} \nabla_\psi \log P(\mathcal{S}|s, \psi) &= \nabla_\psi \left(\log P(s_0 = s) + \sum_{t=0}^{\infty} \log \pi(s_t, a_t, \psi) + \sum_{t=0}^{\infty} \log P_{s_t a_t}^{s_{t+1}} \right) \\ &= \sum_{t=0}^{\infty} \nabla_\psi \log \pi(s_t, a_t, \psi). \end{aligned} \quad (7.16)$$

This is nice, since it implies we do not need the transition model. However, this only holds if the policy is stochastic. If the policy is deterministic we need the gradient $\nabla_\psi \log P_{sa}^s = \nabla_a \log P_{sa}^s \nabla_\psi \pi(s, a, \psi)$, which is available only when the transition probabilities are known. In most cases this is not a big problem, since stochastic policies are needed anyway to ensure sufficient exploration. Figure 7.3 shows two examples of stochastic policies that can be used and the corresponding gradients.

Boltzmann exploration can be used in discrete actions spaces. Assume that $\phi(s,a)$ is a feature vector of size p corresponding to state s and action a . Suppose the policy is a Boltzmann distribution with parameters ψ , such that

$$\pi(s,a,\psi) = \frac{\mathbf{E}^{\psi^T \phi(s,a)}}{\sum_{b \in A(s)} \mathbf{E}^{\psi^T \phi(s,b)}} ,$$

then, the gradient of the logarithm of this policy is given by

$$\nabla_\psi \log \pi(s,a,\psi) = \phi(s,a) - \sum_b \pi(s,b,\psi) \phi(s,b) .$$

Gaussian exploration can be used in continuous action spaces. Consider a Gaussian policy with mean $\mu \in \mathbb{R}^{D_A}$ and $D_A \times D_A$ covariance matrix Σ , such that

$$\begin{aligned} \pi(s,a,\{\mu,\Sigma\}) &= \frac{1}{\sqrt{2\pi \det \Sigma}} \exp\left(-\frac{1}{2}(a-\mu)^T \Sigma^{-1}(a-\mu)\right) , \\ \nabla_\mu \log \pi(s,a,\{\mu,\Sigma\}) &= (a-\mu)^T \Sigma^{-1} , \\ \nabla_\Sigma \log \pi(s,a,\{\mu,\Sigma\}) &= \frac{1}{2} \left(\Sigma^{-1}(a-\mu)(a-\mu)^T \Sigma^{-1} - \Sigma^{-1} \right) . \end{aligned}$$

where the actions $a \in A$ are vectors of the same size as μ . If $\psi \in \Psi \subseteq \mathbb{R}^{D_\mu}$ is a parameter vector that determines the state-dependent location of the mean $\mu(s,\psi)$, then $\nabla_\psi \log \pi(s,a,\psi) = J_\psi^T(\mu(s,\psi)) \nabla_\mu \log \pi(s,a,\{\mu,\Sigma\})$, where $J_\psi(\mu(s,\psi_\mu))$ is the $D_A \times D_\psi$ Jacobian matrix, containing the partial derivatives from each of the elements of $\mu(s,\psi)$ to each of the elements of ψ .

The covariance matrix can be the output of a parametrized function as well, but care should be taken to preserve sufficient exploration. One way is to use natural-gradient updates as normal gradients may decrease the exploration too fast. Another option is to use a covariance matrix $\sigma^2 I$, where σ is a tunable parameter that is fixed or decreased according to some predetermined schedule.

Fig. 7.3 Examples of stochastic policies for policy-gradient algorithms

When we know the gradient in (7.16), we can sample the quantity in (7.15). For this, we need to sample the expected cumulative discounted reward. For instance, if the task is episodic we can take a Monte Carlo sample that gives the cumulative (possibly discounted) reward for each episode. In episodic MDPs, the sum in (7.16) is finite rather than infinite and we obtain

$$\nabla_\psi V^\pi(s_t) = E \left\{ R_k(s_t) \left(\sum_{j=t}^{T_k-1} \nabla_\psi \log \pi(s_j, a_j, \psi) \right) \right\} \quad (7.17)$$

where $R_k(s_t) = \sum_{j=t}^{T_k-1} \gamma^{t-j} r_{j+1}$ is the total (discounted) return obtained after reaching state s_t in episode k , where this episode ended on T_k . This gradient can be sampled and used to update the policy through (7.13).

A drawback of sampling (7.17) is that the variance of $R_k(s_t)$ can be quite high, resulting in noisy estimates of the gradient. Williams (1992) notes that this can be mitigated somewhat by using the following update:

$$\psi_{t+1} = \psi_t + \beta_t(s_t)(R_k(s_t) - b(s_t)) \sum_{j=t}^{T_k} \nabla_\psi \log \pi(s_j, a_j, \psi_t) , \quad (7.18)$$

where $b(s_t)$ is a baseline that does not depend on the policy parameters, although it may depend on the state. This baseline can be used to minimize the variance without adding bias to the update, since for any $s \in S$

$$\begin{aligned} \int_{\mathcal{S}} \nabla_\psi P(\mathcal{S}|s, \psi) b(s) d\mathcal{S} &= b(s) \nabla_\psi \int_{\mathcal{S}} P(\mathcal{S}|s, \psi) d\mathcal{S} \\ &= b(s) \nabla_\psi 1 = 0 . \end{aligned}$$

It has been shown that it can be a good idea to set this baseline equal to an estimate of the state value, such that $b(s) = V_t(s)$ (Sutton et al, 2000; Bhatnagar et al, 2009), although strictly speaking it is then not independent of the policy parameters. Some work has been done to optimally set the baseline to minimize the variance and thereby increase the convergence rate of the algorithm (Greensmith et al, 2004; Peters and Schaal, 2008b), but we will not go into this in detail here.

The policy-gradient updates as defined above all use a gradient that updates the policy parameters in the direction of steepest ascent of the performance metric. However, the gradient update operates in parameter space, rather than in policy space. In other words, when we use normal gradient descent with a step size, we restrict the size of the change in parameter space: $d\psi_t^T d\psi_t$, where $d\psi_t = \psi_{t+1} - \psi_t$ is the change in parameters. It has been argued that it is much better to restrict the step size in policy space. This is similar to our observation in Section 7.2.1.2 that an update in parameter space for a linear function approximator can result in an update in value space with a unintended large or small step size. A good distance metric for policies is the Kullback-Leibler divergence (Kullback and Leibler, 1951; Kullback, 1959). This can be approximated with a second-order Taylor expansion $d\psi_t^T F_\psi d\psi_t$, where F_ψ is the $D_\psi \times D_\psi$ Fisher information matrix, defined as

$$F_\psi = E \left\{ \nabla_\psi P(\mathcal{S}|s, \psi) \nabla_\psi^T P(\mathcal{S}|s, \psi) \right\},$$

where the expectation ranges over the possible trajectories. This matrix can be sampled with use of the identity (7.16). Then, we can obtain a natural policy gradient, which follows a natural gradient (Amari, 1998). This idea was first introduced in reinforcement learning by Kakade (2001). The desired update then becomes

$$\psi_{t+1}^T = \psi_t^T + \beta_t(s_t) F_\psi^{-1} \nabla_\psi V^\pi(s_t) , \quad (7.19)$$

which needs to be sampled. A disadvantage of this update is the need for enough samples to (approximately) compute the inverse matrix F_ψ^{-1} . The number of required samples can be restrictive if the number of parameters is fairly large, especially if a sample consists of an episode that can take many time steps to complete.

Most algorithms that use a natural gradient use $O(D_\psi^2)$ time per update and may require a reasonable amount of samples. More details can be found elsewhere (Kakade, 2001; Peters and Schaal, 2008a; Wierstra et al, 2008; Bhatnagar et al, 2009; Rückstieß et al, 2010).

7.3.2.2 Policy Search with Evolutionary Strategies

Instead of a gradient update on the policy parameters, we can also conduct a gradient-free search in the policy-parameter space. As an interesting example that combines ideas from natural policy-gradients and evolutionary strategies, we discuss natural evolutionary strategies (NES) (Wierstra et al, 2008; Sun et al, 2009). The idea behind the algorithm is fairly straightforward, although many specific improvements are more advanced (Sun et al, 2009). The other gradient-free methods discussed in Section 5 can be used in a similar vein.

Instead of storing a single exploratory policy, NES creates a population of n parameter vectors ψ_1, \dots, ψ_n . These vectors represent policies that have a certain expected payoff. This payoff can be sampled by a Monte Carlo sample $R_k(s_0)$, similar to (7.17), where s_0 is the first state in an episode. This Monte Carlo sample is the fitness. The goal is to improve the population parameters of the distribution that generates the policy parameters, such that the new population distribution will likely yield better policies. In contrast with policy-gradient methods, we do not improve the policies themselves; we improve the process that generates the policies. For this, we use a gradient ascent step on the fitness of the current solutions.

In NES and CMA-ES, the parameter vectors ψ_i are drawn from a Gaussian distribution $\psi_i \sim \mathcal{N}(\mu_\psi, \Sigma_\psi)$. Let ζ_ψ be a vector that contains all the population parameters for the mean and the covariance matrix. NES uses the Monte Carlo samples to find an estimate of the natural gradient $F_\zeta^{-1} \nabla_\zeta E\{R\}$ of the performance to the population parameters in μ_ψ and Σ_ψ . This tells us how the meta-parameters should be changed in order to generate better populations in the future. Because of the choice of a Gaussian generating distribution, it is possible to calculate the Fisher information matrix analytically. With further algorithmic specifics, it is possible to restrict the computation for a single generation in NES to $O(np^3 + nf)$, where n is the number of solutions in the population, p is the number of parameters of a solution and f is the computational cost of determining the fitness for a single solution. Note that f may be large if the necessary Monte Carlo roll-outs can be long. The potentially large variance in the fitness may make direct policy search less appropriate for large, noisy problems. Note that in contrast with policy-gradient algorithms, the candidate policies can be deterministic, which may reduce the variance somewhat.

7.3.2.3 Actor-Critic Algorithms

The variance of the estimate of $\nabla_\psi V^\pi(s_t)$ in (7.17) can be very high if Monte Carlo roll-outs are used, which can severely slow convergence. Likewise, this is a problem for direct policy-search algorithms that use Monte Carlo roll-outs. A potential

solution to this problem is presented by using an explicit approximation of V^π . In this context, such an approximate value function is called a *critic* and the combined algorithm is called an *actor-critic* algorithm (Barto et al, 1983; Sutton, 1984; Konda and Borkar, 1999; Konda, 2002; Konda and Tsitsiklis, 2003).

Actor-critic algorithms typically use a temporal-difference algorithm to update V_t , an estimate for V^π . It can be shown that if a_t is selected according to π , under some assumptions the TD error $\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$ is an unbiased estimate of $Q^\pi(s_t, a_t) - V^\pi(s_t)$. Assuming $b(s_t) = V^\pi(s_t)$ as a baseline, this leads to an unbiased estimate of $\delta_t \nabla_\psi \log \pi(s_t, a_t, \psi_t)$ for the gradient of the policy (Sutton et al, 2000). This estimate can be extended to an approximate natural-gradient direction (Peters et al, 2003; Peters and Schaal, 2008a), leading to natural actor-critic (NAC) algorithms. A typical actor-critic update would update the policy parameters with

$$\psi_{t+1} = \psi_t + \beta_t(s_t) \delta_t \nabla_\psi \log \pi(s_t, a_t, \psi_t) ,$$

where $\nabla_\psi \log \pi(s_t, a_t, \psi_t)$ can be replaced with $F_\psi^{-1} \nabla_\psi \log \pi(s_t, a_t, \psi_t)$ for a NAC algorithm.

In some cases, an explicit approximation of the inverse Fisher information matrix can be avoided by approximating $Q^\pi(s, a) - b(s)$ with a linear function approximator $g_t^\pi(s, a, w) = w_t^T \nabla_\psi \log \pi(s, a, \psi_t)$ (Sutton et al, 2000; Konda and Tsitsiklis, 2003; Peters and Schaal, 2008a). After some algebraic manipulations we then get

$$\nabla_\psi V_t(s) = E\{\nabla_\psi \log \pi(s, a, \psi_t) \nabla_\psi^T \log \pi(s, a, \psi_t)\} w_t = F_\psi w_t ,$$

which we can plug into (7.19) to get the NAC update

$$\psi_{t+1} = \psi_t + \beta_t(s_t) w_t .$$

However, this elegant update only applies to critics that use the specific linear functional form of $g_t^\pi(s, a, w)$ to approximate the value $Q^\pi(s, a) - b(s)$. Furthermore, the accuracy of this update clearly depends on the accuracy of w_t . Other NAC variants are described by Bhatnagar et al (2009).

There is significant overlap between some of the policy-gradient ideas in this section and many of the ideas in the related field of adaptive dynamic programming (ADP) (Powell, 2007; Wang et al, 2009). Essentially, reinforcement learning and ADP can be thought of as different names for the same research field. However, in practice there is a divergence between the sort of problems that are considered and the solutions that are proposed. Usually, in adaptive dynamic programming more of an engineering's perspective is used, which results in a slightly different notation and a somewhat different set of goals. For instance, in ADP the goal is often to stabilize a plant (Murray et al, 2002). This puts some restraints on the exploration that can safely be used and implies that often the goal state is the starting state and the goal is to stay near this state, rather than to find better states. Additionally, problems in continuous time are discussed more often than in reinforcement learning (Beard et al, 1998; Vrabie et al, 2009) for which the continuous version of the Bellman optimality equation is used, that is known as the Hamilton–Jacobi–Bellman equation

(Bardi and Dolcetta, 1997). A further discussion of these specifics falls outside the scope of this chapter.

One of the earliest actor-critic methods stems from the ADP literature. It approximates Q^π , rather than V^π . Suppose we use Gaussian exploration, centered at the output of a deterministic function $Ac : S \times \Psi \rightarrow A$. Here, we refer to this function as the actor, instead of to the whole policy. If we use a differentiable function Q_t to approximate Q^π , it becomes possible to update the parameters of this actor with use of the chain rule:

$$\begin{aligned}\psi_{t+1} &= \psi_t + \alpha_t \nabla_\psi Q_t(s_t, Ac(s, \psi), \theta) \\ &= \psi_t + \alpha_t J_\psi^T(Ac(s_t, \psi)) \nabla_a Q_t(s_t, a) ,\end{aligned}$$

where $J_\psi(Ac(s, \psi))$ is the $D_A \times D_\Psi$ Jacobian matrix of which the element on the i^{th} row and j^{th} column is equal to $\frac{\partial}{\partial \psi_j} Ac_i(s, \psi)$, where $Ac_i(s, \psi)$ is the i^{th} element of $Ac(s, \psi)$. This algorithm is called action dependent heuristic dynamic programming (ADHDP) (Werbos, 1977; Prokhorov and Wunsch, 2002). The critic can be in fact updated with any action-value algorithm, including Q-learning, which would imply an estimate of Q^* rather than Q^π . There are multiple variants of this algorithm, many of which assume a known model of the environment, the reward function or both, or they construct such models. Then, often an additional assumption is that the model is differentiable.

7.3.2.4 Continuous Actor-Critic Learning Automaton

In this section we discuss the continuous actor-critic learning-automaton (Cacla) algorithm (van Hasselt and Wiering, 2007, 2009). In contrast with most other actor-critic methods, Cacla uses an error in action space rather than in parameter or policy space and it uses the sign of the temporal-difference error rather than its size.

In the Cacla algorithm, a critic $V : S \times \Theta \rightarrow \mathbb{R}$ approximates V^π , where π is the current policy. An actor $Ac : S \times \Psi \rightarrow A$ outputs a single—possibly multi-dimensional—action for each state. During learning, it is assumed that there is exploration, such that $a_t \neq Ac(s_t, \psi_t)$ for reasons that will soon become clear. For instance, $\pi(s_t, \psi_t)$ could be a Gaussian distribution centered on $Ac(s_t, \psi_t)$. As in many other actor-critic algorithms, if the temporal-difference error δ_t is positive, we judge a_t to be a good choice and we reinforce it. In Cacla, this is done by updating the output of the actor towards a_t . This is why exploration is necessary: without exploration the actor output is already equal to the action, and the parameters cannot be updated.⁸

An update to the actor only occurs when the temporal-difference error is positive. This is similar to a linear reward-inaction update for learning automata (Narendra

⁸ Feedback on actions equal to the output of the actor can still improve the value function. This can be useful, because then the value function can improve while the actor stays fixed. Similar to policy iteration, we could interleave steps without exploration to update the critic, with steps with exploration to update the actor. Although promising, we do not explore this possibility here further.

```

1: Initialize  $\theta_0$  (below  $V_t(s) = V(s, \theta_t)$ ),  $\psi_0, s_0$ .
2: for  $t \in \{0, 1, 2, \dots\}$  do
3:   Choose  $a_t \sim \pi(s_t, \psi_t)$ 
4:   Perform  $a_t$ , observe  $r_{t+1}$  and  $s_{t+1}$ 
5:    $\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$ 
6:    $\theta_{t+1} = \theta_t + \alpha_t(s_t) \delta_t \nabla_\theta V_t(s_t)$ 
7:   if  $\delta_t > 0$  then
8:      $\psi_{t+1} = \psi_t + \beta_t(s_t)(a_t - Ac(s_t, \psi_t)) \nabla_\psi Ac(s_t, \psi_t)$ 
9:   if  $s_{t+1}$  is terminal then
10:    Reinitialize  $s_{t+1}$ 

```

Algorithm 17. Cacla

and Thathachar, 1974, 1989), using the sign of the temporal-difference error as a measure of ‘success’. Most other actor-critic methods use the size of the temporal-difference error and also update in the opposite direction when its sign is negative. However, this is usually not a good idea for Cacla, since this is equivalent to updating towards some action that was not performed and for which it is not known whether it is better than the current output of the actor. As an extreme case, consider an actor that already outputs the optimal action in each state for some deterministic MDP. For most exploring actions, the temporal-difference error is then negative. If the actor would be updated away from such an action, its output would almost certainly no longer be optimal.

This is an important difference between Cacla and policy-gradient methods: Cacla only updates its actor when actual improvements have been observed. This avoids slow learning when there are plateaus in the value space and the temporal-difference errors are small. It was shown empirically that this can indeed result in better policies than when the step size depends on the size of the temporal-difference error (van Hasselt and Wiering, 2007). Intuitively, it makes sense that the distance to a promising action a_t is more important than the size of the improvement in value.

A basic version of Cacla is shown in Algorithm 17. The policy in line 3 can depend from the actor’s output, but this is not strictly necessary. For instance, unexplored promising parts of the action space could be favored by the action selection. In Section 7.4, we will see that Cacla can even learn from a fully random policy. Cacla can only update its actor when $a_t \neq Ac(s_t, \psi_t)$, but after training has concluded the agent can deterministically use the action that is output by the actor.

The critic update in line 6 is an ordinary TD learning update. One can replace this with a $\text{TD}(\lambda)$ update, an incremental least-squares update or with any of the other updates from Section 7.3.1.2. The actor update in line 8 can be interpreted as gradient descent on the error $\|a_t - Ac(s_t, \psi_t)\|$ between the action that was performed and the output of the actor. This is the second important difference with most other actor-critic algorithms: instead of updating the policy in parameter space (Konda, 2002) or policy space (Peters and Schaal, 2008a; Bhatnagar et al, 2009), we use an error directly in action space.

In some ways, Cacla is similar to an evolutionary strategy. In the context of reinforcement learning, evolutionary strategies usually store a distribution in parameter space, from which policy parameters are sampled. This approach was for instance proposed for NES (Rückstieß et al, 2010), CMA-ES (Heidrich-Meisner and Igel, 2008) and cross-entropy optimization (Buşoniu et al, 2010). Conversely, Cacla uses a probability distribution in action space: the action-selection policy.

Cacla is compatible with more types of exploration than policy-gradient algorithms. For instance, a uniform random policy would still allow Cacla to improve its actor, whereas such a policy has no parameters to tune for policy-gradient methods.

In previous work, Cacla was compared favorably to ADHDP and wire-fitting (van Hasselt and Wiering, 2007) and to discrete temporal-difference methods such as Q-learning and Sarsa (van Hasselt and Wiering, 2009). In the next section, we compare it to CMA-ES and NAC on a double-pole cart pole problem. For simplicity, in the experiment we use the simple version of Cacla outlined in Algorithm 17. However, Cacla can be extended and improved in numerous ways. For completeness, we list some of the possible improvements here.

First, Cacla can be extended with eligibility traces. For instance, the value function can be updated with $\text{TD}(\lambda)$ or the new variants $\text{TDC}(\lambda)$ and $\text{GTD2}(\lambda)$ (Sutton et al, 2009). The latter variants may be beneficial to learn the value function for the actor in an off-policy manner, rather than to learn the value for the stochastic policy that is used for exploration as is done in the simple version of the algorithm. The actor can also be extended with traces that update the actor's output for a certain state a little bit towards the action that was taken there if positive TD errors are observed later. It is not yet clear whether such actor traces improve the performance.

Second, Cacla can be extended with batch updates that make more efficient use of the experiences that were observed in the past. For instance, (incremental) least-squares temporal-difference learning (Bradtko and Barto, 1996; Boyan, 2002; Geramifard et al, 2006) or a variant of (neural) fitted Q-iteration (Ernst et al, 2005; Riedmiller, 2005) can be used. Since this can decrease the variance in the TD errors, this can prevent actor updates to poor actions and may allow for larger actor step sizes. Similarly, samples could be stored in order to reuse them in a form of experience replay (Lin, 1992).

Third, Cacla can use multiple actors. This can prevent the actor from getting stuck in a locally optimal policy. One could then use a discrete algorithm such as Q-learning to choose between the actors. Preliminary results with this approach are promising, although the additional actor selector introduces additional parameters that need to be tuned.

7.4 An Experiment on a Double-Pole Cart Pole

In this section, we compare Cacla, CMA-ES and NAC on a double-pole cart-pole problem. In this problem, two separate poles are attached by a hinge to a cart. The poles differ in length and mass and must both be balanced by hitting the cart.

In reinforcement learning, many different metrics have been used to compare the performance of algorithms and no fully standardized benchmarks exist. Therefore, we compare the results of Cacla to the results for CMA-ES and NAC from an earlier paper (Heidrich-Meisner and Igel, 2008), using the dynamics and the performance metric used therein. We choose this particular paper because it reports results for NAC and for CMA-ES, which is considered the current state-of-the-art in direct policy search and black-box optimization (Jiang et al, 2008; Gomez et al, 2008; Glasmachers et al, 2010; Hansen et al, 2010).

The dynamics of the double cart pole are as follows (Wieland, 1991):

$$\ddot{x} = \frac{F - \mu_c \text{sign}(\dot{x}) + \sum_{i=1}^2 2m_i \dot{\chi}_i^2 \sin \chi_i + \frac{3}{4} m_i \cos \chi_i \left(2 \frac{\mu_i \dot{\chi}_i}{m_i l_i} + g \sin \chi_i \right)}{m_c + \sum_{i=1}^2 m_i \left(1 - \frac{3}{4} \cos^2 \chi_i \right)}$$

$$\ddot{\chi} = -\frac{3}{8l_i} \left(\ddot{x} \cos \chi_i + g \sin \chi_i + 2 \frac{\mu_i \dot{\chi}_i}{m_i l_i} \right)$$

Here $l_1 = 1$ m and $l_2 = 0.1$ m are the lengths of the poles, $m_c = 1$ kg is the weight of the cart, $m_1 = 0.1$ kg and $m_2 = 0.01$ kg are the weights of the poles and $g = 9.81$ m/s² is the gravity constant. Friction is modeled with coefficients $\mu_c = 5 \cdot 10^{-4}$ N s/m and $\mu_1 = \mu_2 = 2 \cdot 10^{-6}$ N m s. The admissible state space is defined by the position of the cart $x \in [-2.4 \text{ m}, 2.4 \text{ m}]$ and the angles of both poles $\chi_i \in [-36^\circ, 36^\circ]$ for $i \in \{1, 2\}$. On leaving the admissible state space, the episode ends. Every time step yields a reward of $r_t = 1$ and therefore it is optimal to make episodes as long as possible. The agent can choose an action from the range $[-50 \text{ N}, 50 \text{ N}]$ every 0.02 s.

Because CMA-ES uses Monte Carlo roll-outs, the task was made explicitly episodic by resetting the environment every 20 s (Heidrich-Meisner and Igel, 2008). This is not required for Cacla, but was done anyway to make the comparison fair. The feature vector is $\phi(s) = (x, \dot{x}, \chi_1, \dot{\chi}_1, \chi_2, \dot{\chi}_2)^T$. All episodes start in $\phi(s) = (0, 0, 1^\circ, 0, 0, 0)^T$. The discount factor in the paper was $\gamma = 1$. This means that the state values are unbounded. Therefore, we use a discount factor of $\gamma = 0.99$. In practice, this makes little difference for the performance. Even though Cacla optimizes the discounted cumulative reward, we use the reward per episode as performance metric, which is explicitly optimized by CMA-ES and NAC.

CMA-ES and NAC were used to train a linear controller, so Cacla is also used to find a linear controller. We use a bias feature that is always equal to one, so we are looking for a parameter vector $\psi \in \mathbb{R}^7$. A hard threshold is used, such that if the output of the actor is larger than 50 N or smaller than -50 N, the agent outputs 50 N or -50 N, respectively. The critic was implemented with a multi-layer perceptron with 40 hidden nodes and a tanh activation function for the hidden layer. The initial controller was initialized with uniformly random parameters between -0.3 and 0.3. No attempt was made to optimize this initial range for the parameters or the number of hidden nodes.

The results by CMA-ES are shown in Figure 7.4. Heidrich-Meisner and Igel (2008) show that NAC performs far worse and therefore we do not show its

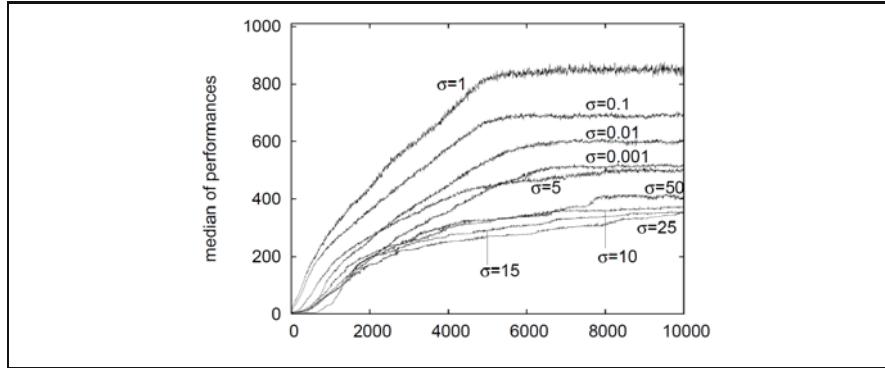


Fig. 7.4 Median reward per episode by CMA-ES out of 500 repetitions of the experiment. The x -axis shows the number of episodes. Figure is taken from Heidrich-Meisner and Igel (2008).

performance. The performance of NAC is better if it is initialized close to the optimal policy, in which case the median performance of NAC reaches the optimal reward per episode of 1000 after 3000 to 4000 episodes. However, this would of course assume a priori knowledge about the optimal solution. The best performance of CMA-ES is a median reward of approximately 850. As shown in the figure, for values of the parameter σ other than $\sigma = 1$, the performance is worse.

We ran Cacla for just 500 episodes with fixed step sizes of $\alpha = \beta = 10^{-3}$ and Gaussian exploration with $\sigma = 5000$. This latter value was coarsely tuned and the reason the exploration is so high is that Cacla effectively learns a bang-bang controller: the resulting actor outputs values far above 50N and far below -50N. The results are very robust to the exact setting of this exploration parameter.

We also ran Cacla with ε -greedy exploration, in which a uniform random action is chosen with probability ε . We used $\varepsilon = 0.1$ and $\varepsilon = 1$, where the latter implies a fully random policy. The ε -greedy versions of Cacla do not learn a bang-bang controller, because the targets for the actor are always within the admissible range. Note that the actor is only updated on average once every ten steps when $\varepsilon = 0.1$, because at the other steps the action is equal to its output.

Table 7.2 shows the results of our experiments with Cacla. The mean reward per episode (with a maximum of 1000) and the success rate is shown, where the latter is the percentage of controllers that can balance the poles for at least 20s. The *online* column shows the average online results for episodes 401–500, including exploration. The *offline* column shows the average offline results, obtained by testing the actor without exploration after 500 episodes were concluded. In Figure 7.4 the median performances are shown, but the online and offline median performance of Cacla with $\sigma = 5000$ and $\varepsilon = 0.1$ is already perfect at 1000. This can be seen from Table 7.2, since in those cases the success rate is over 50%. To be able to compare different exploration techniques for Cacla, we show the mean performances.

Table 7.2 The mean reward per episode (**mean**), the standard error of this mean (**se**) and the percentage of trials where the reward per episode was equal to 1000 (**success**) are shown for Cacla with $\alpha = \beta = 10^{-3}$. Results are shown for training episodes 401–500 (**online**) and for the greedy policy after 500 episodes of training (**offline**). The **action noise** and **exploration** are explained in the main text. Averaged over 1000 repetitions.

action noise	exploration	online			offline		
		mean	se	success	mean	se	success
0	$\sigma = 5000$	946.3	6.7	92.3 %	954.2	6.3	94.5 %
	$\varepsilon = 0.1$	807.6	9.6	59.0 %	875.2	9.4	84.5 %
	$\varepsilon = 1$	29.2	0.0	0 %	514.0	10.4	25.5 %
[-20 N,20 N]	$\sigma = 5000$	944.6	6.9	92.5 %	952.4	6.5	94.5 %
	$\varepsilon = 0.1$	841.0	8.7	60.7 %	909.5	8.1	87.4 %
	$\varepsilon = 1$	28.7	0.0	0 %	454.7	9.5	11.3 %
[-40 N,40 N]	$\sigma = 5000$	936.0	7.4	91.9 %	944.9	7.0	93.8 %
	$\varepsilon = 0.1$	854.2	7.9	50.5 %	932.6	6.7	86.7 %
	$\varepsilon = 1$	27.6	0.0	0 %	303.0	6.7	0 %

On average, the controllers found by Cacla after only 500 episodes are significantly better than those found by CMA-ES after 10,000 episodes. Even $\varepsilon = 1$ results in quite reasonable greedy policies. Naturally, when $\varepsilon = 1$ the online performance is poor, because the policy is fully random. But note that the greedy performance of 514.0 is much better than the performance of CMA-ES after 500 episodes.

To test robustness of Cacla, we reran the experiment with noise in the action execution. A uniform random force in the range $[-20\text{N},20\text{N}]$ or $[-40\text{N},40\text{N}]$ is added to the action before execution. The action noise is added after cropping the actor output to the admissible range and the algorithm is not informed of the amount of added noise. For example, assume the actor of Cacla outputs an action of $Ac(s_t) = 40$. Then Gaussian exploration is added, for instance resulting in $a_t = 70$. This action is not in the admissible range $[-50,50]$, so it is cropped to 50. Then uniform noise, drawn from $[-20,20]$ or $[-40,40]$, is added. Suppose the result is 60. Then, a force of 60N is applied to the cart. If the resulting temporal-difference is positive, the output of the actor for this state is updated towards $a_t = 70$, so the algorithm is unaware of both the cropping and the uniform noise that were applied to its output.

The results including action noise are also shown in Table 7.2. The performance of Cacla is barely affected when Gaussian exploration is used. The slight drop in performance falls within the statistical margins of error, although it does seem consistent. Interestingly, the added noise even improves the online and offline performance of Cacla when ε -greedy exploration with $\varepsilon = 0.1$ is used. Apparently, the added noise results in desirable extra exploration.

This experiment indicates that the relatively simple Cacla algorithm is very effective at solving some continuous reinforcement-learning problems. Other previous work show that natural-gradient and evolutionary algorithms typically need a few

thousand episodes to learn a good policy on the double pole, but also on the single pole task (Sehnke et al, 2010). We do not show the results here, but Cacla also performs very well on the single-pole cart pole. Naturally, this does not imply that Cacla is the best choice for all continuous MDPs. For instance, in a partially observable MDPs an evolutionary approach to directly search in parameter space may find good controllers faster, although it is possible to use Cacla to train a recurrent neural network, for instance with real-time recurrent learning (Williams and Zipser, 1989) or backpropagation through time (Werbos, 2002). Additionally, convergence to an optimal policy or even local optima for variants of Cacla is not (yet) guaranteed, while for some actor-critic (Bhatnagar et al, 2009) and direct policy-search algorithms convergence to a local optimum can be guaranteed.

The reason that Cacla performs much better than CMA-ES on this particular problem is that CMA-ES uses whole episodes to estimate the fitness of a candidate policy and stores a whole population of such policies. Cacla, on the other hand, makes use of the structure of the problem by using temporal-difference errors. This allows it to quickly update its actor, making learning possible even during the first few episodes. NAC has the additional disadvantage that quite a few samples are necessary to make its estimate of the Fisher information matrix accurate enough to find the natural-gradient direction. Finally, the improvements to the actor in Cacla are not slowed down by plateaus in the value space. As episodes become longer, the value space will typically exhibit such plateaus, making the gradient estimates used by NAC more unreliable and the updates smaller. Because Cacla operates directly in action space, it does not have this problem and it can move towards better actions with a fixed step size, whenever the temporal-difference is positive.

As a final note, the simple variant of Cacla will probably not perform very well in problems with specific types of noise. For instance, Cacla may be tempted to update towards actions that often yield fairly high returns but sometimes yield very low returns, making them a poor choice on average. This problem can be mitigated by storing an explicit approximation of the reward function, or by using averaged temporal-difference errors instead of the stochastic errors. These issues have not been investigated in depth.

7.5 Conclusion

There are numerous ways to find good policies in problems with continuous spaces. Three general methodologies exist that differ in which part of the problem is explicitly approximated: the model, the value of a policy, or the policy itself. Function approximation can be used to approximate these functions, which can be updated with gradient-based or gradient-free methods. Many different reinforcement-learning algorithms result from combinations of these techniques. We mostly focused on value-function and policy approximation, because models of continuous MDPs quickly become intractable to solve, making explicit approximations of these less useful.

Several ways to update value-function approximators were discussed, including temporal-difference algorithms such as TD-learning, Q-learning, GTD2 and TDC. To update policy approximators, methods such as policy-gradient, actor-critic and evolutionary algorithms can be used. Because these latter approaches store an explicit approximation for the policy, they can be applied directly to problems where the action space is also continuous.

Of these methods, the gradient-free direct policy-search algorithms have the best convergence guarantees in completely continuous problems. However, these methods can be inefficient, because they use complete Monte Carlo roll-outs and do not exploit the Markov structure of the MDP. Actor-critic methods store both an explicit estimate of the policy and a critic that can exploit this structure, for instance by using temporal differences. These methods have much potential, although they are harder to analyze in general.

To get some idea of the merits of different approaches, the continuous actor-critic learning automaton (Cacla) algorithm (van Hasselt and Wiering, 2007, 2009) was compared on a double-pole balancing problem to the state-of-the-art in black-box optimization and direct policy search: the covariance-matrix adaptation evolution strategy (CMA-ES) (Hansen et al, 2003). In earlier work, CMA-ES was compared favorably to other methods, such as natural evolutionary strategies (NES) (Wierstra et al, 2008; Sun et al, 2009) and the natural actor-critic (NAC) algorithm (Peters and Schaal, 2008a). Our results show that Cacla reaches much better policies in a much smaller number of episodes. A reason for this is that Cacla is an online actor-critic method, whereas the other methods need more samples to deduce the direction to update the policy to. In other words, Cacla uses the available experience samples more efficiently, although it can easily be extended to be even more sample-efficient.

There are less general convergence guarantees in continuous MDPs than in finite MDPs. Some work has been done recently to fill this gap (see, e.g., Bertsekas, 2007; Szepesvári, 2010), but more analysis is still desirable. Many of the current methods are either (partially) heuristic, sample-inefficient or computationally intractable on large problems. However, recent years have shown an increase in theoretical guarantees and practical general-purpose algorithms and we expect this trend will continue. Efficiently finding optimal decision strategies in general problems with large or continuous domains is one of the hardest problems in artificial intelligence, but it is also a topic with many real-world applications and implications.

Acknowledgements. I would like to thank Peter Bosman and the anonymous reviewers for helpful comments.

References

- Akimoto, Y., Nagata, Y., Ono, I., Kobayashi, S.: Bidirectional Relation Between CMA Evolution Strategies and Natural Evolution Strategies. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 154–163. Springer, Heidelberg (2010)
 Albus, J.S.: A theory of cerebellar function. Mathematical Biosciences 10, 25–61 (1971)

- Albus, J.S.: A new approach to manipulator control: The cerebellar model articulation controller (CMAC). In: *Dynamic Systems, Measurement and Control*, pp. 220–227 (1975)
- Amari, S.I.: Natural gradient works efficiently in learning. *Neural Computation* 10(2), 251–276 (1998)
- Anderson, C.W.: Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine* 9(3), 31–37 (1989)
- Antos, A., Munos, R., Szepesvári, C.: Fitted Q-iteration in continuous action-space MDPs. In: *Advances in Neural Information Processing Systems (NIPS-2007)*, vol. 20, pp. 9–16 (2008a)
- Antos, A., Szepesvári, C., Munos, R.: Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning* 71(1), 89–129 (2008b)
- Babuska, R.: *Fuzzy modeling for control*. Kluwer Academic Publishers (1998)
- Bäck, T.: *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA (1996)
- Bäck, T., Schwefel, H.P.: An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation* 1(1), 1–23 (1993)
- Baird, L.: Residual algorithms: Reinforcement learning with function approximation. In: Prieditis, A., Russell, S. (eds.) *Machine Learning: Proceedings of the Twelfth International Conference*, pp. 30–37. Morgan Kaufmann Publishers, San Francisco (1995)
- Baird, L.C., Klopf, A.H.: Reinforcement learning with high-dimensional, continuous actions. Tech. Rep. WL-TR-93-114, Wright Laboratory, Wright-Patterson Air Force Base, OH (1993)
- Bardi, M., Dolcetta, I.C.: Optimal control and viscosity solutions of Hamilton–Jacobi–Bellman equations. Springer, Heidelberg (1997)
- Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics SMC-13*, 834–846 (1983)
- Baxter, J., Bartlett, P.L.: Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research* 15, 319–350 (2001)
- Beard, R., Saridis, G., Wen, J.: Approximate solutions to the time-invariant Hamilton–Jacobi–Bellman equation. *Journal of Optimization theory and Applications* 96(3), 589–626 (1998)
- Bellman, R.: *Dynamic Programming*. Princeton University Press (1957)
- Benbrahim, H., Franklin, J.A.: Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems* 22(3-4), 283–302 (1997)
- Berenji, H.: Fuzzy Q-learning: a new approach for fuzzy dynamic programming. In: *Proceedings of the Third IEEE Conference on Fuzzy Systems, IEEE World Congress on Computational Intelligence*, pp. 486–491. IEEE (1994)
- Berenji, H., Khedkar, P.: Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks* 3(5), 724–740 (1992)
- Bertsekas, D.P.: *Dynamic Programming and Optimal Control*, vol. I. Athena Scientific (2005)
- Bertsekas, D.P.: *Dynamic Programming and Optimal Control*, vol. II. Athena Scientific (2007)
- Bertsekas, D.P., Tsitsiklis, J.N.: *Neuro-dynamic Programming*. Athena Scientific, Belmont (1996)
- Bertsekas, D.P., Borkar, V.S., Nedic, A.: Improved temporal difference methods with linear function approximation. In: *Handbook of Learning and Approximate Dynamic Programming*, pp. 235–260 (2004)

- Beyer, H., Schwefel, H.: Evolution strategies—a comprehensive introduction. *Natural Computing* 1(1), 3–52 (2002)
- Bhatnagar, S., Sutton, R.S., Ghavamzadeh, M., Lee, M.: Natural actor-critic algorithms. *Automatica* 45(11), 2471–2482 (2009)
- Bishop, C.M.: Neural networks for pattern recognition. Oxford University Press, USA (1995)
- Bishop, C.M.: Pattern recognition and machine learning. Springer, New York (2006)
- Bonarini, A.: Delayed reinforcement, fuzzy Q-learning and fuzzy logic controllers. In: Herrera, F., Verdegay, J.L. (eds.) *Genetic Algorithms and Soft Computing. Studies in Fuzziness*, vol. 8, pp. 447–466. Physica-Verlag, Berlin (1996)
- Boyan, J.A.: Technical update: Least-squares temporal difference learning. *Machine Learning* 49(2), 233–246 (2002)
- Bradtko, S.J., Barto, A.G.: Linear least-squares algorithms for temporal difference learning. *Machine Learning* 22, 33–57 (1996)
- Bryson, A., Ho, Y.: *Applied Optimal Control*. Blaisdell Publishing Co. (1969)
- Buşoniu, L., Ernst, D., De Schutter, B., Babuška, R.: Continuous-State Reinforcement Learning with Fuzzy Approximation. In: Tuyls, K., Nowe, A., Guessoum, Z., Kudenko, D. (eds.) ALAMAS 2005, ALAMAS 2006, and ALAMAS 2007. LNCS (LNAI), vol. 4865, pp. 27–43. Springer, Heidelberg (2008)
- Buşoniu, L., Babuška, R., De Schutter, B., Ernst, D.: *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Boca Raton (2010)
- Coulom, R.: Reinforcement learning using neural networks, with applications to motor control. PhD thesis, Institut National Polytechnique de Grenoble (2002)
- Crites, R.H., Barto, A.G.: Improving elevator performance using reinforcement learning. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) *Advances in Neural Information Processing Systems*, vol. 8, pp. 1017–1023. MIT Press, Cambridge (1996)
- Crites, R.H., Barto, A.G.: Elevator group control using multiple reinforcement learning agents. *Machine Learning* 33(2/3), 235–262 (1998)
- Davis, L.: *Handbook of genetic algorithms*. Arden Shakespeare (1991)
- Dayan, P.: The convergence of $\text{TD}(\lambda)$ for general lambda. *Machine Learning* 8, 341–362 (1992)
- Dayan, P., Sejnowski, T.: $\text{TD}(\lambda)$: Convergence with probability 1. *Machine Learning* 14, 295–301 (1994)
- Dearden, R., Friedman, N., Russell, S.: Bayesian Q-learning. In: Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, pp. 761–768. American Association for Artificial Intelligence (1998)
- Dearden, R., Friedman, N., Andre, D.: Model based Bayesian exploration. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp. 150–159 (1999)
- Eiben, A.E., Smith, J.E.: *Introduction to evolutionary computing*. Springer, Heidelberg (2003)
- Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6(1), 503–556 (2005)
- Fisher, R.A.: On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London Series A, Containing Papers of a Mathematical or Physical Character* 222, 309–368 (1922)
- Fisher, R.A.: Statistical methods for research workers. Oliver & Boyd, Edinburgh (1925)
- Främling, K.: Replacing eligibility trace for action-value learning with function approximation. In: Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN-2007), pp. 313–318. d-side publishing (2007)

- Gaskett, C., Wettergreen, D., Zelinsky, A.: Q-learning in continuous state and action spaces. In: Advanced Topics in Artificial Intelligence, pp. 417–428 (1999)
- Geramifard, A., Bowling, M., Sutton, R.S.: Incremental least-squares temporal difference learning. In: Proceedings of the 21st National Conference on Artificial Intelligence, vol. 1, pp. 356–361. AAAI Press (2006)
- Geramifard, A., Bowling, M., Zinkevich, M., Sutton, R.: ilstd: Eligibility traces and convergence analysis. In: Advances in Neural Information Processing Systems, vol. 19, pp. 441–448 (2007)
- Glasmachers, T., Schaul, T., Yi, S., Wierstra, D., Schmidhuber, J.: Exponential natural evolution strategies. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 393–400. ACM (2010)
- Glorennec, P.: Fuzzy Q-learning and dynamical fuzzy Q-learning. In: Proceedings of the Third IEEE Conference on Fuzzy Systems, IEEE World Congress on Computational Intelligence, pp. 474–479. IEEE (1994)
- Glover, F., Kochenberger, G.: Handbook of metaheuristics. Springer, Heidelberg (2003)
- Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. *The Journal of Machine Learning Research* 9, 937–965 (2008)
- Gordon, G.J.: Stable function approximation in dynamic programming. In: Priditidis, A., Russell, S. (eds.) Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995), pp. 261–268. Morgan Kaufmann, San Francisco (1995)
- Gordon, G.J.: Approximate solutions to Markov decision processes. PhD thesis, Carnegie Mellon University (1999)
- Greensmith, E., Bartlett, P.L., Baxter, J.: Variance reduction techniques for gradient estimates in reinforcement learning. *The Journal of Machine Learning Research* 5, 1471–1530 (2004)
- Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9(2), 159–195 (2001)
- Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation* 11(1), 1–18 (2003)
- Hansen, N., Auger, A., Ros, R., Finck, S., Pošík, P.: Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In: Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO 2010, pp. 1689–1696. ACM, New York (2010)
- Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice Hall PTR (1994)
- Heidrich-Meisner, V., Igel, C.: Evolution Strategies for Direct Policy Search. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 428–437. Springer, Heidelberg (2008)
- Holland, J.H.: Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)* 9(3), 297–314 (1962)
- Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
- Howard, R.A.: Dynamic programming and Markov processes. MIT Press (1960)
- Huyer, W., Neumaier, A.: SNOBFIT–stable noisy optimization by branch and fit. *ACM Transactions on Mathematical Software (TOMS)* 35(2), 1–25 (2008)
- Jiang, F., Berry, H., Schoenauer, M.: Supervised and Evolutionary Learning of Echo State Networks. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 215–224. Springer, Heidelberg (2008)

- Jouffe, L.: Fuzzy inference system learning by reinforcement methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 28(3), 338–355 (1998)
- Kakade, S.: A natural policy gradient. In: Dietterich, T.G., Becker, S., Ghahramani, Z. (eds.) *Advances in Neural Information Processing Systems* 14 (NIPS-2001), pp. 1531–1538. MIT Press (2001)
- Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: *Proceedings of IEEE International Conference on Neural Networks*, Perth, Australia, vol. 4, pp. 1942–1948 (1995)
- Kirkpatrick, S.: Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics* 34(5), 975–986 (1984)
- Klir, G., Yuan, B.: *Fuzzy sets and fuzzy logic: theory and applications*. Prentice Hall PTR, Upper Saddle River (1995)
- Konda, V.: Actor-critic algorithms. PhD thesis, Massachusetts Institute of Technology (2002)
- Konda, V.R., Borkar, V.: Actor-critic type learning algorithms for Markov decision processes. *SIAM Journal on Control and Optimization* 38(1), 94–123 (1999)
- Konda, V.R., Tsitsiklis, J.N.: Actor-critic algorithms. *SIAM Journal on Control and Optimization* 42(4), 1143–1166 (2003)
- Kullback, S.: *Statistics and Information Theory*. J. Wiley and Sons, New York (1959)
- Kullback, S., Leibler, R.A.: On information and sufficiency. *Annals of Mathematical Statistics* 22, 79–86 (1951)
- Lagoudakis, M., Parr, R.: Least-squares policy iteration. *The Journal of Machine Learning Research* 4, 1107–1149 (2003)
- Lin, C., Lee, C.: Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems. *IEEE Transactions on Fuzzy Systems* 2(1), 46–63 (1994)
- Lin, C.S., Kim, H.: CMAC-based adaptive critic self-learning control. *IEEE Transactions on Neural Networks* 2(5), 530–533 (1991)
- Lin, L.: Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8(3), 293–321 (1992)
- Lin, L.J.: Reinforcement learning for robots using neural networks. PhD thesis, Carnegie Mellon University, Pittsburgh (1993)
- Littman, M.L., Szepesvári, C.: A generalized reinforcement-learning model: Convergence and applications. In: Saitta, L. (ed.) *Proceedings of the 13th International Conference on Machine Learning* (ICML 1996), pp. 310–318. Morgan Kaufmann, Bari (1996)
- Maei, H.R., Sutton, R.S.: GQ (λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In: *Proceedings of the Third Conference On Artificial General Intelligence* (AGI-2010), pp. 91–96. Atlantis Press, Lugano (2010)
- Maei, H.R., Szepesvári, C., Bhatnagar, S., Precup, D., Silver, D., Sutton, R.: Convergent temporal-difference learning with arbitrary smooth function approximation. In: *Advances in Neural Information Processing Systems* 22 (NIPS-2009) (2009)
- Maei, H.R., Szepesvári, C., Bhatnagar, S., Sutton, R.S.: Toward off-policy learning control with function approximation. In: *Proceedings of the 27th Annual International Conference on Machine Learning* (ICML-2010). ACM, New York (2010)
- Maillard, O.A., Munos, R., Lazaric, A., Ghavamzadeh, M.: Finite sample analysis of Bellman residual minimization. In: *Asian Conference on Machine Learning*, ACML-2010 (2010)
- Mitchell, T.M.: *Machine learning*. McGraw Hill, New York (1996)
- Moriarty, D.E., Miikkulainen, R.: Efficient reinforcement learning through symbiotic evolution. *Machine Learning* 22, 11–32 (1996)
- Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research* 11, 241–276 (1999)

- Murray, J.J., Cox, C.J., Lendaris, G.G., Saeks, R.: Adaptive dynamic programming. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 32(2), 140–153 (2002)
- Narendra, K.S., Thathachar, M.A.L.: Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics* 4, 323–334 (1974)
- Narendra, K.S., Thathachar, M.A.L.: Learning automata: an introduction. Prentice-Hall, Inc., Upper Saddle River (1989)
- Nedić, A., Bertsekas, D.P.: Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems* 13(1-2), 79–110 (2003)
- Neyman, J., Pearson, E.S.: On the use and interpretation of certain test criteria for purposes of statistical inference part i. *Biometrika* 20(1), 175–240 (1928)
- Ng, A.Y., Parr, R., Koller, D.: Policy search via density estimation. In: Solla, S.A., Leen, T.K., Müller, K.R. (eds.) *Advances in Neural Information Processing Systems*, vol. 13, pp. 1022–1028. The MIT Press (1999)
- Nguyen-Tuong, D., Peters, J.: Model learning for robot control: a survey. *Cognitive Processing*, 1–22 (2011)
- Ormoneit, D., Sen, Š.: Kernel-based reinforcement learning. *Machine Learning* 49(2), 161–178 (2002)
- Pazis, J., Lagoudakis, M.G.: Binary action search for learning continuous-action control policies. In: *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 793–800. ACM (2009)
- Peng, J.: Efficient dynamic programming-based learning for control. PhD thesis, Northeastern University (1993)
- Peters, J., Schaal, S.: Natural actor-critic. *Neurocomputing* 71(7-9), 1180–1190 (2008a)
- Peters, J., Schaal, S.: Reinforcement learning of motor skills with policy gradients. *Neural Networks* 21(4), 682–697 (2008b)
- Peters, J., Vijayakumar, S., Schaal, S.: Reinforcement learning for humanoid robotics. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids 2003)*. IEEE Press (2003)
- Poupart, P., Vlassis, N., Hoey, J., Regan, K.: An analytic solution to discrete Bayesian reinforcement learning. In: *Proceedings of the 23rd International Conference on Machine Learning*, pp. 697–704. ACM (2006)
- Powell, M.: UOBYQA: unconstrained optimization by quadratic approximation. *Mathematical Programming* 92(3), 555–582 (2002)
- Powell, M.: The NEWUOA software for unconstrained optimization without derivatives. In: *Large-Scale Nonlinear Optimization*, pp. 255–297 (2006)
- Powell, W.B.: *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley-Blackwell (2007)
- Precup, D., Sutton, R.S.: Off-policy temporal-difference learning with function approximation. In: *Machine Learning: Proceedings of the Eighteenth International Conference (ICML 2001)*, pp. 417–424. Morgan Kaufmann, Williams College (2001)
- Precup, D., Sutton, R.S., Singh, S.P.: Eligibility traces for off-policy policy evaluation. In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pp. 766–773. Morgan Kaufmann, Stanford University, Stanford, CA (2000)
- Prokhorov, D.V., Wunsch, D.C.: Adaptive critic designs. *IEEE Transactions on Neural Networks* 8(5), 997–1007 (2002)
- Puterman, M.L.: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York (1994)

- Puterman, M.L., Shin, M.C.: Modified policy iteration algorithms for discounted Markov decision problems. *Management Science* 24(11), 1127–1137 (1978)
- Rao, C.R., Poti, S.J.: On locally most powerful tests when alternatives are one sided. *Sankhyā: The Indian Journal of Statistics*, 439–439 (1946)
- Rechenberg, I.: *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog (1971)
- Riedmiller, M.: Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 317–328. Springer, Heidelberg (2005)
- Ripley, B.D.: *Pattern recognition and neural networks*. Cambridge University Press (2008)
- Rubinstein, R.: The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability* 1(2), 127–190 (1999)
- Rubinstein, R., Kroese, D.: The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning. Springer-Verlag New York Inc. (2004)
- Rückstieß, T., Sehnke, F., Schaul, T., Wierstra, D., Sun, Y., Schmidhuber, J.: Exploring parameter space in reinforcement learning. *Paladyn* 1(1), 14–24 (2010)
- Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: *Parallel Distributed Processing*, vol. 1, pp. 318–362. MIT Press (1986)
- Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. *Tech. Rep. CUED/F-INFENG-TR 166*, Cambridge University, UK (1994)
- Santamaria, J.C., Sutton, R.S., Ram, A.: Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior* 6(2), 163–217 (1997)
- Scherrer, B.: Should one compute the temporal difference fix point or minimize the Bellman residual? The unified oblique projection view. In: Fürnkranz, J., Joachims, T. (eds.) *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pp. 959–966. Omnipress (2010)
- Schwefel, H.P.: *Numerische Optimierung von Computer-Modellen. Interdisciplinary Systems Research*, vol. 26. Birkhäuser, Basel (1977)
- Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., Schmidhuber, J.: Parameter-exploring policy gradients. *Neural Networks* 23(4), 551–559 (2010)
- Singh, S.P., Sutton, R.S.: Reinforcement learning with replacing eligibility traces. *Machine Learning* 22, 123–158 (1996)
- Spaan, M., Vlassis, N.: Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research* 24(1), 195–220 (2005)
- Stanley, K.O., Miikkulainen, R.: Efficient reinforcement learning through evolving neural network topologies. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pp. 569–577. Morgan Kaufmann, San Francisco (2002)
- Strehl, A.L., Li, L., Wiewiora, E., Langford, J., Littman, M.L.: PAC model-free reinforcement learning. In: *Proceedings of the 23rd International Conference on Machine Learning*, pp. 881–888. ACM (2006)
- Strens, M.: A Bayesian framework for reinforcement learning. In: *Proceedings of the Seventeenth International Conference on Machine Learning*, p. 950. Morgan Kaufmann Publishers Inc. (2000)
- Sun, Y., Wierstra, D., Schaul, T., Schmidhuber, J.: Efficient natural evolution strategies. In: *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation (GECCO-2009)*, pp. 539–546. ACM (2009)

- Sutton, R.S.: Temporal credit assignment in reinforcement learning. PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci. (1984)
- Sutton, R.S.: Learning to predict by the methods of temporal differences. *Machine Learning* 3, 9–44 (1988)
- Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) *Advances in Neural Information Processing Systems*, vol. 8, pp. 1038–1045. MIT Press, Cambridge (1996)
- Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*. The MIT press, Cambridge (1998)
- Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: *Advances in Neural Information Processing Systems 13 (NIPS-2000)*, vol. 12, pp. 1057–1063 (2000)
- Sutton, R.S., Szepesvári, C., Maei, H.R.: A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. In: *Advances in Neural Information Processing Systems 21 (NIPS-2008)*, vol. 21, pp. 1609–1616 (2008)
- Sutton, R.S., Maei, H.R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., Wiewiora, E.: Fast gradient-descent methods for temporal-difference learning with linear function approximation. In: *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, pp. 993–1000. ACM (2009)
- Szepesvári, C.: Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 4(1), 1–103 (2010)
- Szepesvári, C., Smart, W.D.: Interpolation-based Q-learning. In: *Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2004)*, p. 100. ACM (2004)
- Szita, I., Lörincz, A.: Learning tetris using the noisy cross-entropy method. *Neural Computation* 18(12), 2936–2941 (2006)
- Taylor, M.E., Whiteson, S., Stone, P.: Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, p. 1328. ACM (2006)
- Tesauro, G.: Practical issues in temporal difference learning. In: Lippman, D.S., Moody, J.E., Touretzky, D.S. (eds.) *Advances in Neural Information Processing Systems*, vol. 4, pp. 259–266. Morgan Kaufmann, San Mateo (1992)
- Tesauro, G.: TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation* 6(2), 215–219 (1994)
- Tesauro, G.J.: Temporal difference learning and TD-Gammon. *Communications of the ACM* 38, 58–68 (1995)
- Thrun, S., Schwartz, A.: Issues in using function approximation for reinforcement learning. In: Mozer, M., Smolensky, P., Touretzky, D., Elman, J., Weigend, A. (eds.) *Proceedings of the 1993 Connectionist Models Summer School*. Lawrence Erlbaum, Hillsdale (1993)
- Touzet, C.F.: Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems* 22(3/4), 251–281 (1997)
- Tsitsiklis, J.N., Van Roy, B.: An analysis of temporal-difference learning with function approximation. Tech. Rep. LIDS-P-2322, MIT Laboratory for Information and Decision Systems, Cambridge, MA (1996)
- Tsitsiklis, J.N., Van Roy, B.: An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control* 42(5), 674–690 (1997)
- van Hasselt, H.P.: Double Q-Learning. In: *Advances in Neural Information Processing Systems*, vol. 23. The MIT Press (2010)
- van Hasselt, H.P.: Insights in reinforcement learning. PhD thesis, Utrecht University (2011)

- van Hasselt, H.P., Wiering, M.A.: Reinforcement learning in continuous action spaces. In: Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-2007), pp. 272–279 (2007)
- van Hasselt, H.P., Wiering, M.A.: Using continuous action spaces to solve discrete problems. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN 2009), pp. 1149–1156 (2009)
- van Seijen, H., van Hasselt, H.P., Whiteson, S., Wiering, M.A.: A theoretical and empirical analysis of Expected Sarsa. In: Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning, pp. 177–184 (2009)
- Vapnik, V.N.: The nature of statistical learning theory. Springer, Heidelberg (1995)
- Vrabie, D., Pastravanu, O., Abu-Khalaf, M., Lewis, F.: Adaptive optimal control for continuous-time linear systems based on policy iteration. *Automatica* 45(2), 477–484 (2009)
- Wang, F.Y., Zhang, H., Liu, D.: Adaptive dynamic programming: An introduction. *IEEE Computational Intelligence Magazine* 4(2), 39–47 (2009)
- Watkins, C.J.C.H.: Learning from delayed rewards. PhD thesis, King's College, Cambridge, England (1989)
- Watkins, C.J.C.H., Dayan, P.: Q-learning. *Machine Learning* 8, 279–292 (1992)
- Werbos, P.J.: Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University (1974)
- Werbos, P.J.: Advanced forecasting methods for global crisis warning and models of intelligence. In: General Systems, vol. XXII, pp. 25–38 (1977)
- Werbos, P.J.: Backpropagation and neurocontrol: A review and prospectus. In: IEEE/INNS International Joint Conference on Neural Networks, Washington, D.C, vol. 1, pp. 209–216 (1989a)
- Werbos, P.J.: Neural networks for control and system identification. In: Proceedings of IEEE/CDC, Tampa, Florida (1989b)
- Werbos, P.J.: Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks* 2, 179–189 (1990)
- Werbos, P.J.: Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE* 78(10), 1550–1560 (2002)
- Whiteson, S., Stone, P.: Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research* 7, 877–917 (2006)
- Whitley, D., Dominic, S., Das, R., Anderson, C.W.: Genetic reinforcement learning for neurocontrol problems. *Machine Learning* 13(2), 259–284 (1993)
- Wieland, A.P.: Evolving neural network controllers for unstable systems. In: International Joint Conference on Neural Networks, vol. 2, pp. 667–673. IEEE, New York (1991)
- Wiering, M.A., van Hasselt, H.P.: The QV family compared to other reinforcement learning algorithms. In: Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning, pp. 101–108 (2009)
- Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural evolution strategies. In: IEEE Congress on Evolutionary Computation (CEC-2008), pp. 3381–3387. IEEE (2008)
- Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 229–256 (1992)
- Williams, R.J., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* 1(2), 270–280 (1989)
- Wilson, D.R., Martinez, T.R.: The general inefficiency of batch training for gradient descent learning. *Neural Networks* 16(10), 1429–1451 (2003)
- Zadeh, L.: Fuzzy sets. *Information and Control* 8(3), 338–353 (1965)
- Zhou, C., Meng, Q.: Dynamic balance of a biped robot using fuzzy reinforcement learning agents. *Fuzzy Sets and Systems* 134(1), 169–187 (2003)

Chapter 8

Solving Relational and First-Order Logical Markov Decision Processes: A Survey

Martijn van Otterlo

Abstract. In this chapter we survey representations and techniques for Markov decision processes, reinforcement learning, and dynamic programming in worlds explicitly modeled in terms of *objects* and *relations*. Such relational worlds can be found everywhere in planning domains, games, real-world indoor scenes and many more. Relational representations allow for expressive and natural datastructures that capture the objects and relations in an explicit way, enabling generalization over objects and relations, but also over similar problems which differ in the number of objects. The field was recently surveyed completely in (van Otterlo, 2009b), and here we describe a large portion of the main approaches. We discuss model-free – both value-based and policy-based – and model-based dynamic programming techniques. Several other aspects will be covered, such as models and hierarchies, and we end with several recent efforts and future directions.

8.1 Introduction to Sequential Decisions in Relational Worlds

As humans we speak and think about the world as being made up of *objects* and *relations* among objects. There are books, tables and houses, tables *inside* houses, books *on top of* tables, and so on. “*We are equipped with an inductive bias, a predisposition to learn to divide the world up into objects, to study the interaction of those objects, and to apply a variety of computational modules to the representation of these objects*” (Baum, 2004, p. 173). For intelligent agents this should not be different. In fact, “*...it is hard to imagine a truly intelligent agent that does not conceive of the world in terms of objects and their properties and relations to other objects*” (Kaelbling et al, 2001). Furthermore, such representations are highly

Martijn van Otterlo
Radboud University Nijmegen, The Netherlands
e-mail: m.vanotterlo@donders.ru.nl

effective and compact: "The description of the world in terms of objects and simple interactions is an enormously compressed description" (Baum, 2004, p. 168).

The last decade a new subfield in reinforcement learning (RL) has emerged that tries to endow intelligent agents with both the knowledge representation of (probabilistic) first-order logic – to deal with objects and relations – and efficient RL algorithms – to deal with decision-theoretic learning in complex and uncertain worlds. This field – **relational RL** (van Otterlo, 2009b) – takes inspiration, representational methodologies and algorithms from diverse fields (see Fig. 8.2(left)) such as RL (Sutton and Barto, 1998), *logic-based artificial intelligence* (Minker, 2000), knowledge representation (KR) (Brachman and Levesque, 2004), *planning* (see Russell and Norvig, 2003), *probabilistic-logical machine learning* (De Raedt, 2008) and *cognitive architectures* (Langley, 2006). In many of these areas the use of objects and relations is widespread and it is the assumption that if RL is to be applied in these fields – e.g. for cognitive agents learning to behave, or RL in tasks where communicating the acquired knowledge is required – one has to investigate how RL algorithms interact with expressive formalisms such as first-order logic. Relational RL offers a new representational paradigm for RL and can tackle many problems more compactly and efficiently than state-of-the-art *propositional* approaches, and in addition, can tackle new problems that could not be handled before. It also offers new opportunities to inject additional *background knowledge* into algorithms, surpassing *tabula rasa* learning.

In this chapter we introduce relational RL as new representational paradigm in RL. We start by introducing the elements of generalization in RL and briefly sketch historical developments. In Section 8.2 we introduce relational representations for Markov decision processes (MDPs). In Sections 8.3 and 8.4 we survey model-based and model-free solution algorithms. In addition we survey other approaches, such as hierarchies and model learning, in Section 8.5, and we describe recent developments in Section 8.6. We end with conclusions and future directions in Section 8.7.

8.1.1 MDPs: Representation and Generalization

Given MDP $M = \langle S, A, T, R \rangle$ the usual goal is to compute value functions (V or Q) or directly compute a policy π . For this, a plethora of algorithms exists which all more or less fall in the category of *generalized policy iteration* (GPI) (see Sutton and Barto, 1998), which denotes the iterative loop involving **i**) *policy evaluation*, computing V^π , and **ii**) *policy improvement*, computing an improved policy π' from V^π . GPI is formulated for opaque states and actions, not taking into account the use of representation (and generalization).

In (van Otterlo, 2009b) we formulated a more general concept PIAGET, in which we make the use of representation explicit (see Fig. 8.1). It distinguishes four ways in which GPI interacts with representation. PIAGET-0 is about first generating a compact representation (an *abstraction level*), e.g. through *model minimization*, or *feature extraction*. PIAGET-1 and PIAGET-2 are closest to GPI in that they

assume a *fixed* representation, and learn either MDP-related functions (Q , V) over this abstraction level (PIAGET-1) or they learn *parameters* (e.g. neural network weights) of the representation simultaneously (PIAGET-2). PIAGET-3 methods are the most general; they *adapt* abstraction levels while solving an MDP.

Since relational RL is above all a representational upgrade of RL we have to look at concepts of generalization (or abstraction) in MDPs. We can distinguish five types typically used in MDPs: **i)** state space abstraction, **ii)** factored MDP representations, **iii)** value functions, **iv)** policies, and **v)** hierarchical decompositions. Relational RL focuses on representing an MDP in terms of objects and relations, and then employing *logical generalization* to obtain any of the five types, possibly making use of results obtained with propositional representations.

Generalization is closely tied to representation. In other words, *one can only learn what one can represent*. Even though there are many ways of representing knowledge, including *rules*, *neural networks*, *entity-relationship models*, *first-order logic* and so on, there are few *representational classes*:

Atomic. Most descriptions of algorithms and proofs use so-called atomic state representations, in which the state space S is a discrete *set*, consisting of discrete states s_1 to s_N and A is a set of actions. No abstraction (generalization) is used, and all computation and storage (e.g. of values) happens *state-wise*.

Propositional. This is the most common form of MDP representation (see Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998; Boutilier et al, 1999, for thorough descriptions). Each state consists of a *vector of attribute-value pairs*, where each attribute (*feature*, *random variable*) represents a *measurable quantity* of the domain. Features can be binary or real-valued. Usually the action set A is still a discrete set. Propositional encodings enable the use of, e.g. *decision trees* or *support vector machines* to represent value functions and policies, thereby generalizing over parts of the state space.

Deictic. Some works use *deictic* representations (e.g. Finney et al, 2002) in which propositional encodings are used to *refer* to objects. For example, the value of the feature "*the color of the object in my hand*" could express something about a specific object, without explicitly naming it. Deictic representations, as well as *object-oriented* representations (Diuk et al, 2008) are still propositional, but bridge the gap to explicit relational formalisms.

Relational. A relational representation is an expressive *knowledge representation* (KR) format in which *objects* and *relations* between objects can be expressed in an explicit way, and will be discussed in the next section.

Each representational class comes with its own ways to generalize and abstract. Atomic states do not offer much in that respect, but for propositional representations

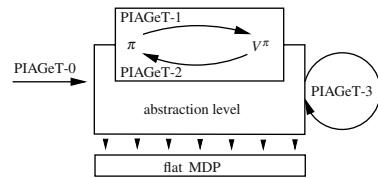


Fig. 8.1 Policy Iteration using Abstraction and Generalization Techniques (PIAGET)

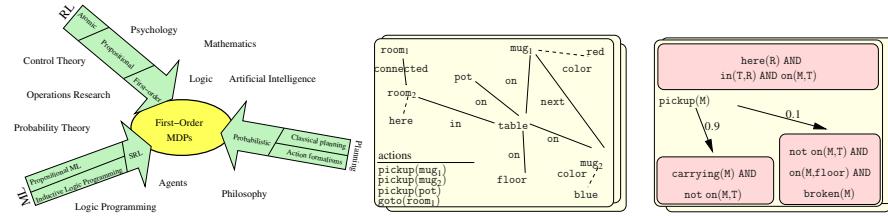


Fig. 8.2 (left) The material in this chapter embedded in various subfields of artificial intelligence (AI), **(middle)** relational representation of a scene, **(right)** generalization over a probabilistic action

much is possible. Generalization makes use of *structure* inherent in problem domains. Such structure can be found and exploited in two distinct ways. One can *exploit structure in representations*, making solutions compact, but also *employ structured representations in algorithms*, making RL algorithms more efficient.

8.1.2 Short History and Connections to Other Fields

The research described in this chapter is a natural development in at least three areas. The core element (see Fig. 8.2) is a *first-order* (or *relational*) MDP.

Reinforcement learning. Before the mid-nineties, many *trial-and-error learning*, *optimal control*, *dynamic programming* (DP) and *value function approximation* techniques were developed. The book by Sutton and Barto (1998) marked the beginning of a new era in which more sophisticated algorithms and representations were developed, and more theory was formed concerning approximation and convergence. Around the turn of the millennium, relational RL emerged as a new subfield, going beyond the propositional representations used until then. Džeroski et al (1998) introduced a first version of Q-learning in a relational domain, and Boutilier et al (2001) reported the first value iteration algorithm in first-order logic. Together they initiated a new *representational* direction in RL.

Machine Learning. Machine learning (ML), of which RL is a subfield, has used relational representations much earlier. *Inductive logic programming* (ILP) (Bergadano and Gunetti, 1995) has a long tradition in learning logical concepts from data. Still much current ML research uses purely propositional representations and focuses on probabilistic aspects (Alpaydin, 2004). However, the last decade logical and probabilistic approaches are merging in the field of *statistical relational learning* (SRL) (De Raedt, 2008). Relational RL itself can be seen as an additional step, adding a *utility framework* to SRL.

Action languages. Planning and cognitive architectures are old AI subjects, but before relational RL surprisingly few approaches could deal efficiently with decision-theoretic concepts. First-order action languages ranging from STRIPS (Fikes and Nilsson, 1971) to *situation calculus* (Reiter, 2001) were primarily

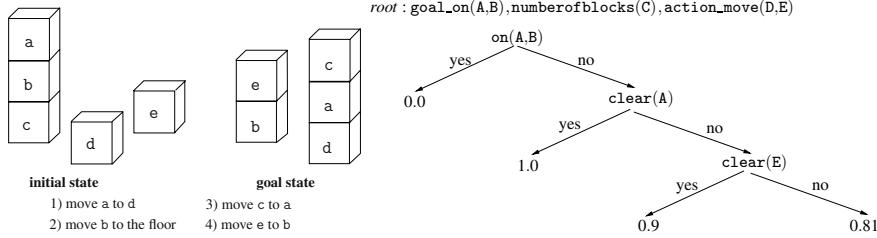


Fig. 8.3 (left) a Blocks World planning problem and an optimal plan, **(right)** a logical Q -tree

used in deterministic, goal-based planning contexts, and the same holds for cognitive architectures. Much relational RL is based on existing action formalisms, thereby extending their applicability to solve decision-theoretic problems.

8.2 Extending MDPs with Objects and Relations

Here we introduce relational representations and logical generalization, and how to employ these for the formalization of *relational* (or, *first-order*) MDPs. Our description will be fairly informal; for more detailed treatment see (van Otterlo, 2009b) and for logical formalizations in general see (Brachman and Levesque, 2004).

8.2.1 Relational Representations and Logical Generalization

Objects and relations require a formalism to express them in explicit form. Fig. 8.2 (**middle**) depicts a typical indoor scene with several *objects* such as `room1`, `pot` and `mug2`. *Relations* between objects are the solid lines, e.g. the relations `connected(room1, room2)` and `on(table, floor)`. Some relations merely express *properties* of objects (i.e. the dashed lines) and can be represented as `color(mug1, red)`, or as a *unary* relation `red(mug1)`. Actions are represented in a similar way, rendering them *parameterized*, such as `pickup(mug1)` and `goto(room1)`.

Note that a propositional representation of the scene would be cumbersome. Each relation between objects should be represented by a binary feature. In order to represent a state by a feature *vector*, all objects and relations should first be fixed, and ordered, to generate the features. For example, it should contain `on_mug1_table = 1` and `on_mug2_table = 1`, but also useless features such as `on_table_table = 0` and `on_room1_mug2 = 0`. This would blow up in size since it must contain many irrelevant relations (many of which do not hold in a state), and would be very inflexible

if the number of objects or relations would vary. Relational representations can naturally deal with these issues.

An important way to generalize over objects is by using *variables* that can stand for different objects (denoted by uppercase letters). For example, in Fig. 8.2(**right**) an action specification makes use of that in order to pickup an object denoted by M. It specifies that the object should be on some object denoted by T, in the same current location R, and M could be e.g. mug₁ or mug₂. The bottom two rectangles show two different *outcomes* of the action, governed by a probability distribution.

Out of many artificial domains, the Blocks World (Slaney and Thiébaux, 2001, and Fig. 8.3(**left**)) is probably the most well-known problem in areas such as KR and planning (see Russell and Norvig, 2003; Schmid, 2001; Brachman and Levesque, 2004) and recently, relational RL. It is a computationally hard problem for general purpose AI systems, it has a relatively simple form, and it supports meaningful, systematic and affordable experiments. Blocks World is often used in relational RL and we use it here to explain important concepts.

More generally, a relational representation consists of a *domain of objects* (or, *constants*) C, and a set of relations {p/α} (or, *predicates*) of which each can have several arguments (i.e. the *arity* α). A *ground atom* is a relation applied to constants from C, e.g. on(a,b). For generalization, usually a *logical language* is used which – in addition to the domain C and set of predicates P – also contains *variables*, *quantifiers* and *connectives*. In the sentence (i.e. formula) $Z \equiv \exists X, Y \text{ on}(X, Y) \wedge \text{clear}(X)$ in such a language, Z represents all states in which there are (expressed by the quantifier \exists) two objects (or, blocks) X and Y which are on each other, and also (expressed by the connective \wedge , or, the *logical AND*) that X is clear. In Fig. 8.3(**left**), in the initial state X could only be a, but in the goal state, it could refer to c or e. Note that Z is not ground since it contains variables.

Logical abstractions can be *learned* from data. This is typically done through methods of ILP (Bergadano and Gunetti, 1995) or SRL (De Raedt, 2008). The details of these approaches are outside the scope of this chapter. It suffices here to see them as *search algorithms* through a vast space of logical abstractions, similar in spirit to propositional tree and rule learners (Alpaydin, 2004). The structure of the space is given by the expressivity of the logical language used. In case of SRL, an additional problem is learning the *parameters* (e.g. probabilities).

8.2.2 Relational Markov Decision Processes

We first need to represent all the aspects of the *problem* in relational form, i.e. MDPs in relational form, based on a domain of objects D and a set of predicates P.

Definition 8.2.1. Let $P = \{p_1/\alpha_1, \dots, p_n/\alpha_n\}$ be a set of predicates with their arities, $C = \{c_1, \dots, c_k\}$ a set of constants, and let $A' = \{a_1/\alpha_1, \dots, a_m/\alpha_m\}$ be a set of actions with their arities. Let S' be the set of all ground atoms that can be constructed from P and C, and let A be the set of all ground atoms over A' and C.

A **relational Markov decision process (RMDP)** is a tuple $M = \langle S, A, T, R \rangle$, where S is a subset of $2^{S'}$, A is defined as stated, $T :: S \times A \times S \rightarrow [0,1]$ is a probabilistic transition function and $R :: S \times A \times S \rightarrow \mathbb{R}$ a reward function.

Thus, each state is represented by the set of ground relational atoms that are true in that state (called an *interpretation* in logic, or a *ground state* here), actions are ground relational atoms, and the transition and reward function are defined as usual, but now over ground relational states and actions. This means that, unlike propositional representations, states are now *unordered* sets of relational atoms and the number of atoms is not fixed.

Example 8.2.1 Using an example of a five-block world based on the predicate set $P = \{\text{on}/2, \text{cl}/1\}$, the constant set $C = \{\text{a}, \text{b}, \text{c}, \text{d}, \text{e}, \text{floor}\}$ and the action set $A' = \{\text{move}/2\}$ we can define the blocks world containing 5 blocks with $|S| = 501$ legal states. T can be defined such that it complies with the standard dynamics of the Blocks World move operator (possibly with some probability of failure) and R can be set positive for states that satisfy some goal criterium and 0 otherwise.

A concrete state s_1 is $\{\text{on}(\text{a}, \text{b}), \text{on}(\text{b}, \text{c}), \text{on}(\text{c}, \text{d}), \text{on}(\text{d}, \text{e}), \text{on}(\text{e}, \text{floor}), \text{cl}(\text{a})\}$ in which all blocks are stacked. The action $\text{move}(\text{a}, \text{floor})$ moves the top block a on the floor. The resulting state s_2 would be $\{\text{on}(\text{a}, \text{floor}), \text{cl}(\text{a}), \text{on}(\text{b}, \text{c}), \text{on}(\text{c}, \text{d}), \text{on}(\text{d}, \text{e}), \text{on}(\text{e}, \text{floor}), \text{cl}(\text{b})\}$ unless there is a probability that the action fails, in which case the current state stays s_1 . If the goal would be to stack all blocks, state reaching s_1 would get a positive reward and s_2 0.

8.2.3 Abstract Problems and Solutions

Relational MDPs form the core model underlying all work in relational RL. Usually they are not posed in ground form, but specified by logical languages which can differ much in their expressivity (e.g. which quantifiers and connectives they support) and reasoning complexity. Here we restrict our story to a simple form: an **abstract state** is a conjunction $Z \equiv Z_1 \wedge \dots \wedge Z_m$ of logical atoms, which can contain variables. A conjunction is implicitly *existentially quantified*, i.e. an abstract state $Z_1 \equiv \text{on}(X, Y)$ should be seen as $\exists X \exists Y Z_1$ which reads as *there are two blocks, denoted X and Y AND block X is on block Y*. For brevity we will omit the quantifiers and connectors from now. An abstract state Z *models* a ground state z if we can find a *substitution* of the variables in Z such that all the substituted atoms appear in z . It is said that Z θ -*subsumes* z (with θ the substitution). For example, Z_1 models (subsumes) the state $\text{clear}(\text{a}), \text{on}(\text{a}, \text{b}), \text{clear}(\text{c}), \dots$ since we can substitute X and Y with a and b and then $\text{on}(\text{a}, \text{b})$ appears in z . Thus, an abstract state generalizes over a *set* of states of the underlying RMDP, i.e. it is an *aggregate* state.

Abstract states are also (partially) ordered by a *θ -subsumption*. If an abstract state Z_1 subsumes another abstract state Z_2 , then Z_1 is *more general* than Z_2 . A *domain theory* supports reasoning and constraint handling to check whether states are legal (wrt. to the underlying semantics of the domain) and to *extend* states (i.e. derive new facts based on the domain theory). This could for the *Blocks World* allow to derive

from $\text{on}(a,b)$ that also $\text{under}(b,a)$ is true. In addition, it would also exclude states that are easy to generate from relational atoms, but are impossible in the domain, e.g. $\text{on}(a,a)$, using the rule $\text{on}(X,X) \rightarrow \text{false}$.

The transition function in Definition 8.2.1 is usually defined in terms of *abstract actions*. Their definition depends on a specific *action logic* and many different forms exists. An action induces a mapping between abstract states, i.e. from a *set* of states to another set of states. An abstract action defines a *probabilistic* action operator by means of a probability distribution over a set of deterministic action outcomes. A generic definition is the following, based on *probabilistic STRIPS* (Hanks and McDermott, 1994).

$$\begin{array}{l} \text{c1}(X), \text{c1}(Y), \text{on}(X,Z), \xrightarrow[X \neq Y, Y \neq Z, X \neq Z]{ 0.9 : \text{move}(X,Y) } \text{on}(X,Y), \text{c1}(X), \text{c1}(Z), \\ \text{c1}(X), \text{c1}(Y), \text{on}(X,Z), \xrightarrow[X \neq Y, Y \neq Z, X \neq Z]{ 0.1 : \text{move}(X,Y) } \text{c1}(X), \text{c1}(Y), \text{on}(X,Z), \\ \quad X \neq Y, Y \neq Z, X \neq Z. \end{array} \quad (8.1)$$

which moves block X on Y with probability 0.9. With probability 0.1 the action fails, i.e., we do not change the state. For an action rule $\text{pre} \rightarrow \text{post}$, if pre is θ -subsumed by a state z , then in the resulting state z' is z with $\text{pre } \theta$ (applied substitution) removed, and $\text{post } \theta$ added. Applied to $z \equiv \text{c1}(a), \text{c1}(b), \text{on}(a,c)$ the action tells us that $\text{move}(a,b)$ will result in $z' \equiv \text{on}(a,b), \text{c1}(a), \text{c1}(c)$ with probability 0.9 and with probability 0.1 we stay in z . An action defines a probabilistic mapping over the set of state-action-state pairs.

Abstract (state) **reward functions** are easy to specify with abstract states. A generic definition of R is the set $\{\langle Z_i, r \rangle \mid i = 1 \dots N\}$ where each Z_i is an abstract state and r is a numerical value. Then, for each relational state z of an RMDP, we can define the reward of z to be the reward given to that abstract state that generalizes over z , i.e. that subsumes z . If rewards should be unique for states (not additive) one has to ensure that R forms a partition over the complete state space of the RMDP.

Overall, by specifying **i**) a domain C and a set of relations P , **ii**) an abstract action definition, and **iii**) an abstract reward function, one can define RMDPs in a compact way. By only changing the domain C one obtains RMDP variants consisting of different sets of objects, i.e. a *family* of related RMDPs (see van Otterlo, 2009b, for more details).

For solution elements – policies and value functions – we can employ similar representations. In fact, although value functions are different from reward functions, we can use the same representation in terms of a set of abstract states with values. However, other – more compact – representations are possible. To represent a state-action **value function**, Džeroski et al (1998) employed a *relational decision tree*, i.e. a compact partition of the state-action space into regions of the same value. Fig. 8.3(right) depicts such a tree, where the root node specifies the action $\text{move}(D,E)$. All state-action pairs that try to move block D on E , in a state where a block A is not on B and where A is *clear*, get a value 1. Note that, in contrast to propositional trees, node tests share variables.

Policies are mappings from states to actions, and they are often represented using relational decision lists (Mooney and Califf, 1995). For example, consider the following *abstract* policy for a Blocks World, which optimally encodes how to reach states where $\text{on}(a,b)$ holds:

```

r1 : move(X,floor) ← onTop(X,b)
r2 : move(Y,floor) ← onTop(Y,a)
r3 : move(a,b)      ← clear(a),clear(b)
r4 : noop          ← on(a,b)

```

where *noop* denotes doing nothing. Rules are read from top to bottom: given a state, the first rule where the abstract state applies, generates an optimal action.

Relational RL, either model-free or model-based, has to cope with *representation generation* and *control learning*. A starting point is the RMDP with associated value functions V and Q and policy π that correspond to the problem, and the overall goal of learning is an (optimal) abstract policy $\tilde{\Pi}$.

$$\text{RMDP } M = \langle S, A, T, R \rangle \xrightarrow{\text{control learning + representation learning}} \tilde{\Pi}^* : S \rightarrow A \quad (8.2)$$

Because learning the policy in the ground RMDP is not an option, various routes can be taken in order to find $\tilde{\Pi}^*$. One can first construct *abstract value functions* and deduce a policy from that. One might also inductively learn the policy from optimal ground traces. Relational abstraction over RMDPs induces a number of *structural* learning tasks. In the next sections we survey various approaches.

8.3 Model-Based Solution Techniques

Model-based algorithms rely on the assumption that a full (abstract) model of the RMDP is available, such that it can be used in DP algorithms to compute value functions and policies. A characteristic solution pattern is the following series of purely deductive steps:

$$\begin{array}{ccccccc} \tilde{V}^0 \equiv \mathcal{R} & \xrightarrow{\mathbf{D}} & \tilde{V}^1 & \xrightarrow{\mathbf{D}} & \tilde{V}^2 & \xrightarrow{\mathbf{D}} & \dots \xrightarrow{\mathbf{D}} \tilde{V}^k & \xrightarrow{\mathbf{D}} & \tilde{V}^{k+1} & \xrightarrow{\mathbf{D}} & \dots \\ \downarrow \mathbf{D} & & \downarrow \mathbf{D} & & \downarrow \mathbf{D} & & & & \downarrow \mathbf{D} & & \downarrow \mathbf{D} \\ \tilde{\Pi}^0 & & \tilde{\Pi}^1 & & \tilde{\Pi}^2 & & & & \tilde{\Pi}^k & & \tilde{\Pi}^{k+1} \end{array} \quad (8.3)$$

The initial specification of the problem can be viewed as a *logical theory*. The initial reward function \mathcal{R} is used as the initial zero-step value function \tilde{V}^0 . Each subsequent abstract value function \tilde{V}^{k+1} is obtained from \tilde{V}^k by *deduction* (\mathbf{D}). From each value function \tilde{V}^k a policy $\tilde{\Pi}^k$ can be deduced. At the end of the section we will briefly discuss additional sample-based approaches. Relational model-based solution algorithms generally make explicit use of *Bellman optimality equations*. By turning these into *update rules* they then heavily utilize the KR capabilities to exploit *structure* in both *solutions* and *algorithms*. This has resulted in several relational versions of traditional algorithms such as *value iteration* (VI).

8.3.1 The Structure of Bellman Backups

Let us take a look at the following Bellman update rule , which is usually applied to all states $s \in S$ simultaneously in an iteration:

$$V^{k+1}(s) = (\mathbf{B}^* V^k)(s) = R(s) + \gamma \max_a \sum_{s' \in S} T(s,a,s') V^k(s') \quad (8.4)$$

When used in VI, it extends a k -steps-to-go horizon of the value function V^k to a $(k+1)$ -steps-to-go value function V^{k+1} using the *backup operator* \mathbf{B}^* . When viewed in more detail, we can distinguish the following sub-operations that together compute the new value $V^{k+1}(s)$ of state $s \in S$:

1. **Matching/Overlap:** First we look at the states s' that are reachable by some action, for which we know the value $V^k(s')$.
2. **Regression:** The value of s is computed by *backing up the value* of a state that can be reached from s by performing an action a . Equivalently, if we know $V^k(s')$ and we know for which states s , $T(s,a,s') > 0$ then we can infer a *partial Q-value* $Q(s,a) = \gamma \cdot T(s,a,s') \cdot V^k(s')$ by reasoning "backwards" from s' to s .
3. **Combination:** Since an action a can have multiple, probabilistic outcomes (i.e. transitions to other states) when applied in state s , the partial Q -values are to be combined by summing them, resulting in "the" Q -value $Q(s,a) = \sum_{s' \in S} T(s,a,s') V^k(s')$. Afterwards the reward $R(s)$ can be added.
4. **Maximization:** In each iteration the highest state value is wanted, and therefore the final value $V^{k+1}(s)$ maximizes over the set of applicable actions.

In a similar way algorithms such as *policy iteration* can be studied.

Intensional Dynamic Programming

Now, classical VI computes values *individually* for each state, even though many states will share the exact same transition pattern to other states. Structured (relational) representations can be used to avoid such redundancy, and compute values for many states simultaneously. Look again at the abstract action definition in Equation 8.1. This compactly specifies many transition probabilities in the form of rules. In a similar way, abstract value functions can represent compactly the values for a set of states using just a single abstract state. **Intensional dynamic programming** (IDP) (van Otterlo, 2009a) makes the use of KR formalisms explicit in MDPs, and provides a unifying framework for structured DP. IDP is defined *representation-independent* but can be instantiated with any atomic, propositional or relational representation. The core of IDP consists of expressing the four mentioned computations (overlap, regression, combination, and maximization) by *representation-dependent* counterparts. Together they are called *decision-theoretic regression* (DTR). A simple instantiation of IDP is *set-based* DP, in which value functions are represented

as discrete sets, and DP employs *set-based* backups. Several other algorithms in the literature can be seen as instantiations of IDP in the propositional context.

A first development in IDP is *explanation-based RL* (EBRL) (Dietterich and Flann, 1997). The inspiration for this work is the similarity between Bellman backups and *explanation-based generalization* (EBG) in *explanation-based learning* (EBL) (see Minton et al, 1989). The representation that is used consists of propositional rules and *region-based* (i.e. for 2D grid worlds) representations. Boutilier et al (2000) introduced *structured* value and policy iteration algorithms (SVI and SPI) based on *propositional trees* for value functions and policies and *dynamic Bayesian networks* for representing the transition function. Later these techniques were extended to work with even more compact *algebraic decision diagrams* (ADD). A third instantiation of IDP are the (hyper)-rectangular partitions of continuous state spaces (Feng et al, 2004). Here, so-called *rectangular piecewise-constant* (RPWC) functions are stored using *kd-trees* (splitting hyperplanes along k axes) for efficient manipulation. IDP techniques are conceptually easy to extend towards POMDPs (Boutilier and Poole, 1996). Let us now go to the relational setting.

8.3.2 Exact Model-Based Algorithms

Exact model-based algorithms for RMDPs implement the four components of DTR in a specific *logical* formalism. The starting point is an abstract action as specified in Equation 8.1. Let us assume our reward function R , which is the initial value function V^0 , is $\{\langle \text{on}(a,b), \text{on}(c,d), 10 \rangle, \langle \text{true}, 0 \rangle\}$. Now, in the first step of DTR, we can use the action specification to find abstract states to backup the values in R to. Since the value function is not specified in terms of individual states, we can use **regression** to find out *what are the conditions for the action in order to end up in an abstract state*. Let us consider the first outcome of the action (with probability 0.9), and take $\text{on}(a,b), \text{on}(c,d)$ in the value function V^0 . **Matching** now amounts to checking whether there is an *overlap* between states expressed by $\text{on}(X,Y), \text{c1}(X), \text{c1}(Z)$ and $\text{on}(a,b), \text{on}(c,d)$. That is, we want to consider those states that are modeled by both the action effect and the part the value function we are looking at. It turns out there are four ways of computing this overlap (see Fig. 8.4) due to the use of variables (we omit constraints here).

Let us pick the first possibility in the picture, then what happened was that $\text{on}(X,Y)$ was matched against $\text{on}(a,b)$, which says that we consider the case where the action performed has *caused* a being on b. Matching has generated the substitutions X/a and Y/b which results in the matched state being $Z' \equiv \text{on}(a,b), \text{c1}(a), \text{c1}(Z)$ "plus" $\text{on}(c,d)$ which is the part the action did not cause. Now the regression step is *reasoning backwards* through the action, and finding out what the possible abstract state should have been in order for the action $\text{move}(a,b)$ to have caused Z' . This is a straightforward computation and results in $Z \equiv \text{on}(a,Z), \text{c1}(b)$ "plus" $\text{on}(c,d)$. Now, in terms of the DTR algorithm, we can compute a partial Q-value $Q(Z, \text{move}(a,b)) = \gamma \cdot T(Z, \text{move}(a,b), Z') \cdot V^k(Z') = \gamma \cdot 0.9 \cdot 10$.

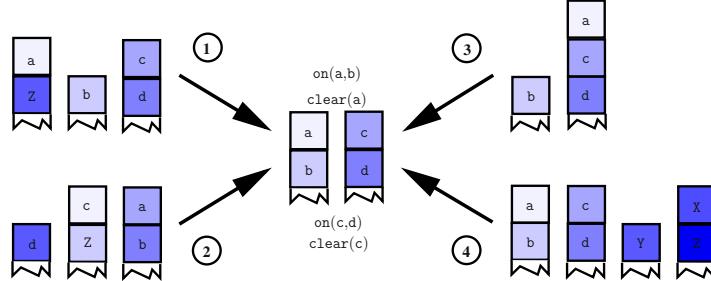


Fig. 8.4 Examples of regression in a Blocks World. All four configurations around the center state are possible ‘pre-states’ when regressing the center state through a standard move action. The two left configurations are situations in which either block a is put onto b (1), or block c is put onto d (2). Block Z in both these configurations can be either a block or the floor. In the top right configuration (3) block a was on block c and is now put onto b . The fourth configuration (4) is interesting because it assumes that none of the blocks a , b , c , d is actually moved, but that there are additional blocks which were moved (i.e. blocks X and Y).

Doing these steps for all actions, all action outcomes and all abstract states in the value function V^k results in a set of abstract state-action pairs $\langle z, a, Q \rangle$ representing partial Q -values. The **combination** of partial Q -values into a Q -function is again done by computing an overlap, in this case between states appearing in the partial Q -function. Let us assume we have computed another partial Q -value $Q(\mathbb{Z}^2, \text{move}(a,b)) = 3$ for $\mathbb{Z}^2 \equiv \text{on}(a,Z), \text{c1}(b), \text{on}(c,d), \text{on}(e,f)$, now for the second outcome of the action (with probability 0.1). Now, in the overlap state $\mathbb{Z}^3 \equiv \mathbb{Z} \wedge \mathbb{Z}^2 \equiv \text{on}(a,Z), \text{c1}(b), \text{on}(c,d), \text{on}(e,f)$ we know that doing action $\text{move}(a,b)$ in state \mathbb{Z}^3 has an expected value of $10 + 3$. The combination of partial Q -values has to be performed for all possible combinations of action outcomes, for all actions.

The last part of DTR is the **maximization**. In our example case, this is fairly easy, since the natural subsumption order on abstract states does most of the work. One can just sort the Q -function $\{\langle \mathbb{Z}, \mathbb{A}, Q \rangle\}$ and take for any state-action pair the first rule that subsumes it. For efficiency reasons, one can remove rules in the Q -function that are subsumed by a higher-valued rule. For other formalisms, the maximization step sometimes involves an extra reasoning effort to maximize over different variable substitutions. The whole procedure can be summarized in the following algorithm, called *first-order decision-theoretic regression* (FODTR):

Require: an abstract value function V^n

- 1: **for each** action type $\mathbb{A}(\mathbb{X})$ **do**
- 2: compute $Q_{V^k}^{\mathbb{A}(\mathbb{X})} = R \oplus [\gamma \otimes \bigoplus_j (\text{prob}(\mathbb{A}(\mathbb{X})) \otimes \text{Regr}(V^k, \mathbb{A}_j(\mathbb{X})))]$
- 3: $Q_{V^k}^{\mathbb{A}} = \text{obj-max}(Q_{V^k}^{\mathbb{A}(\mathbb{X})})$
- 4: $V^{k+1} = \max_{\mathbb{A}} Q_{V^k}^{\mathbb{A}}$

where \otimes and \oplus denote multiplication and summation over Cartesian products of abstraction-value pairs, Regr denotes a regression operator, and obj-max denotes the maximization over Q -functions.

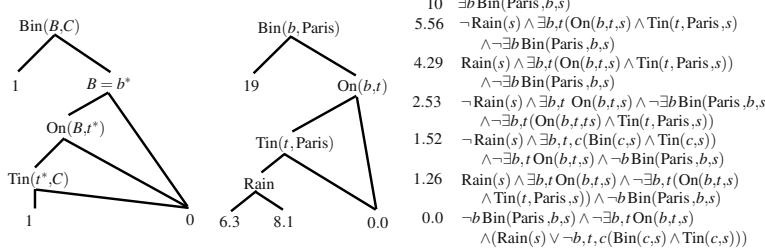


Fig. 8.6 Examples of structures in exact, first-order IDP. Both are concrete examples using the logistics domain described in our experimental section. The left two figures are first-order decision diagrams taken from (Wang et al, 2007). On the far left, the transition function (or, *truth value diagram*) is depicted for $\text{Bin}(B,C)$ under action choice $\text{unload}(b^*, t^*)$. The right diagram depicts the value function V^1 , which turns out to be equivalent to Q_{unload}^1 . The formulas on the right represent the final value partition for the logistics domain computed in (Boutilier et al, 2001).

Currently, four published versions of exact, first-order IDP algorithms have appeared in the literature. The first method is the *symbolic dynamic programming* (SDP) approach (Boutilier et al, 2001) based on the *situation calculus* language (see Reiter, 2001). The second approach, FOVI (Hölldobler and Skvortsova, 2004), is based on the *fluent calculus* (Thielscher, 1998). The third approach is ReBel (Kersting et al, 2004). The most recent approach is based on *first-order decision diagrams* (see Groote and Tveretina, 2003), and will be called FODD here (Wang et al, 2008a).

SDP was the first system but it came without a practical implementation. An important reason was the expressivity of the logic used in SDP, which makes all operations in FODTR computationally expensive. The other three approaches have been shown to be computationally feasible, though at the expense of lower expressivity. Fig. 8.5 shows a computed optimal value function for the goal $\text{on}(a,b)$ for a 10-block world (which amounts to almost 60 million ground states). Both ReBel and FOVI operate on conjunctive states with limited negation, whereas FODTR uses very compact data structures. This comes with the price of being more expensive in terms of keeping the representation small. Just like its propositional counterpart (SVI) the FODTR procedure in FODD operates on the entire value function, whereas ReBel and FOVI (and SDP) work at the level of individual abstract states. Transition functions in all formalisms are based on the underlying logic; probabilistic STRIPS rules for

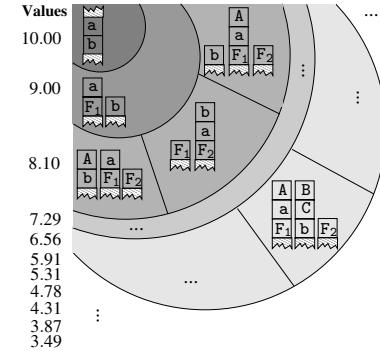


Fig. 8.5 Blocks World abstract value function. Parts of the abstract value function for $\text{on}(a,b)$ after 10 iterations (values rounded). It contains just over a hundred rules. F_i can be a block or a floor block. States more than 10 steps away from the goal get value 0.0.

ReBel, decision diagrams in FODD, and both SDP and FOVI use their underlying fluent and situation calculus action specifications. Figure 8.6 shows some examples of representations used in SDP and FODD. Note how exact, but also how complex, a simple value function in SDP becomes. FODD’s decision diagrams are highly compact, but strictly less expressive than SDP state descriptions. *Regression* is built-in as a main reasoning method in SDP and FOVI but for ReBel and FODD specialized procedures were invented.

All four methods perform IDP in first-order domains, *without first grounding the domain*, thereby computing solutions directly on an abstract level. On a slightly more general level, FODTR can be seen as a means to perform (*lifted*) *first-order* reasoning over decision-theoretic values, i.e. as a kind of decision-theoretic logic. One can say that all four methods *deduce* optimal utilities of states (possibly in infinite state spaces) through FODTR, using the action definitions and domain theory as a set of axioms.

Several extensions to the four systems have been described, for example, *policy extraction* and the use of *tabling* (van Otterlo, 2009b), search-based exploration and efficient subsumption tests (Karabaev et al, 2006), *policy iteration* (Wang and Khordon, 2007), *factored* decomposition of first-order MDPs and *additive reward models* (Sanner and Boutilier, 2007), and universally quantified goals (Sanner and Boutilier, 2006).

8.3.3 Approximate Model-Based Algorithms

Since exact, optimal value functions are complex to compute and store (and may be even infinite, in case of an unlimited domain size), several works have developed approximate model-based algorithms for RMDPs. The first type of approach starts from FODTR approaches and then approximates, and a second type uses other means, for example sampling and planning.

The *first-order approximate linear programming* technique (FOALP) (Sanner and Boutilier, 2005) extends the SDP approach, transforming it into an *approximate* value iteration (AVI) algorithm (Schuurmans and Patrascu, 2001). Instead of exactly representing the complete value function, which can be large and fine-grained (and because of that hard to compute), the authors use a fixed set of *basis functions*, comparable to abstract states. That is, a value function can be represented as a *weighted sum of k first-order basis functions* each containing a *small* number of formulae that provide a first-order abstraction (i.e. partition) of the state space. The backup of a linear combination of such basis functions is simply the linear combination of the FODTR of each basis function. Unlike exact solutions where value functions can grow exponentially in size and where much effort goes into logical simplification of formulas, this feature-based approach must only look for good weights for the case statements. Related to FOALP, the *first-order approximate policy iteration* algorithm (FOAPI) (Sanner and Boutilier, 2006) is a first-order generalization of

approximate policy iteration for factored MDPs (e.g. see Guestrin et al, 2003b). It uses the same basis function decomposition of value functions as the FOALP approach, and in addition, an explicit policy representation. It iterates between two phases. In the first, the value function for the current policy is computed, i.e. the weights of the basis functions are computed using LP. The second phase computes the policy from the value function. Convergence is reached if the policy remains stable between successive approximations. Loss bounds for the converged policy generalize directly from the ones for factored MDPs. The PRM approach by Guestrin too provides bounds on policy quality. Yet, these are PAC-bounds obtained under the assumption that the probability of domains falls off exponentially with their size. The FOALP bounds on policy quality apply equally to all domains. FOALP was used for *factored* FOMDPs by Sanner and Boutilier (2007) and applied in the SysAdmin domain. Both FOALP and FOAPI have been entered the *probabilistic* part of the *international planning competition* (IPPC). In a similar AVI framework as FOALP, Wu and Givan (2007) describe a technique for generating first-order features. A simple ILP algorithm employs a *beam-search* in the feature space, guided by how well each feature correlates with the ideal Bellman residual.

A different approach is the approximation to the SDP approach described by Gretton and Thiébaux (2004a,b). The method uses the same basic setup as SDP, but the FODTR procedure is only partially computed. By employing multi-step *classical* regression from the goal states, a number of structures is computed that represent abstract states. The combination and maximization steps are not performed, but instead the structures generated by regression are used as a *hypothesis language* in the higher-order inductive tree-learner ALKEMY (Lloyd, 2003) to induce a tree representing the value function.

A second type of approximate algorithm does not function at the level of abstraction (as in FODTR and extensions) but uses sampling and generalization in the process of generating solutions. That is, one can first generate a solution for one or more (small) ground instances (plans, value functions, policies), and then use inductive generalization methods to obtain generalized solutions. This type of solution was pioneered by the work of Lecoeuche (2001) who used a solved instance of an RMDP to obtain a generalized policy in a dialogue system. Two other methods that use complete, ground RMDP solutions are based on *value function* generalization (Mausam and Weld, 2003) and *policy* generalization (Cocora et al, 2006). Both approaches first use a general MDP solver and both use a relational decision tree algorithm to generalize solutions using relatively simple logical languages. de la Rosa et al (2008) also use a relational decision tree in the ROLLER algorithm to learn *generalized* policies from examples generated by a heuristic planner. The *Relational Envelope-Based Planning* (REBP) (Gardiol and Kaelbling, 2003) uses a representation of a limited part of the state space (the envelope) which is gradually expanded through sampling just outside the envelope. The aim is to compute a policy, by first generating a good initial plan and use envelope-growing to improve the robustness of the plans incrementally. A more recent extension of the

method allows for the representation of the envelope using varying numbers of predicates, such that the representational complexity can be gradually increased during learning (Gardiol and Kaelbling, 2008).

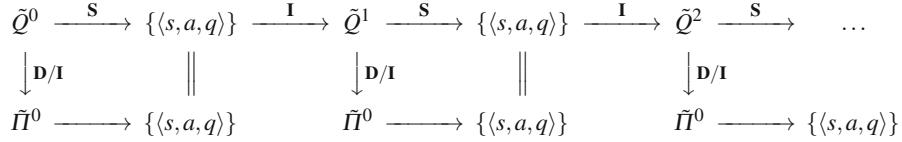
In the following table we summarize the approaches in this section.

Table 8.1 Main model-based approaches that were discussed. Legend: BW=Blocks World, Cnj=conjunction of logical atoms, Q=Q-learning, PS=prioritized sweeping, E=exact, A=approximate, PP-IPC=planning problems from the planning contest (IPC), FOF=first-order (relational) features, PRM=probabilistic relational model, FOL=first-order logic.

method	representation	type	algorithm	applications
IDP (van Otterlo, 2009b)	any	E	IDP	—
SDP (Boutilier et al, 2001)	FOL	E	IDP	logistics
ReBel (Kersting et al, 2004)	Conj	E	IDP	BW, logistics
FOVI (Hölldobler and Skvortsova, 2004)	Conj	E	IDP	BW
FODD (Wang et al, 2008a)	FO-ADD	E	IDP	BW
FODD (Wang and Khordon, 2007)	FO-ADD	E	IDP/API	BW
(Guestrin, 2003)	PRM	A	sampling	FreeCraft, SysAdmin
FOALP (Sanner and Boutilier, 2005)	FOF	A	AVI	elevator
FOAPI (Sanner and Boutilier, 2006)	FOF	A	API	PP-IPC
(Gretton and Thiébaux, 2004a)	adaptive FOF	A	FODTR/VFA	logistics, BW
(Wu and Givan, 2007)	adaptive FOF	A	Bellman residuals	PP-IPC
FOLAO* (Karabaev and Skvortsova, 2005)	adaptive FOF	A	FO-LAO*	PP-IPC
REBP (Gardiol and Kaelbling, 2003)	envelopes	A	planning	BW
adaptive REBP (Gardiol and Kaelbling, 2008)	envelopes	A	planning	BW

8.4 Model-Free Solutions

Here we review techniques that do not assume availability of an abstract model of the underlying RMDP. Many approaches use the following typical Q -learning pattern:



Here, an initial abstract Q -function is used to get (**S**) biased learning samples from the RMDP. The samples are then used to learn (or, *induce* (**I**)) a new Q -function structure. Policies can be computed (or, *deduced* (**D**)) from the current Q -function. A restricted variation on this scheme, discussed in the next section, is to fix the logical abstraction level (e.g. \tilde{Q}) and only sample the RMDP to get good estimates of the values (e.g. parameters of \tilde{Q}).

8.4.1 Value-Function Learning with Fixed Generalization

Several techniques use relational formalisms to create a compact representation of the underlying RMDP, fix that abstraction level, and then apply standard value learning algorithms such as Q-learning. An example abstraction for a three-block Blocks World in the CARCASS representation (van Otterlo, 2003, 2004) is:

```

state ( $\$_1$ ):      (on(A,B),on(B,floor),on(C,floor),A ≠ B,B ≠ C)
actions ( $A_{11}, \dots, A_{13}$ ): move(A,C),move(C,A),move(A,floor)
state ( $\$_2$ ):      (on(A,floor),on(B,floor),on(C,floor), A ≠ B,B ≠ C,A ≠ C)
actions ( $A_{21}, \dots, A_{26}$ ): move(A,B),move(B,A),move(A,C),move(C,A),move(B,C),move(C,B)
state ( $\$_3$ ):      (on(A,B),on(B,C),on(C,floor),A ≠ B,B ≠ C,C ≠ floor)
actions ( $A_{31}$ ): move(A,floor)

```

It transforms the underlying RMDP into a much smaller abstract MDP, which can then be solved by (modifications of) RL algorithms. For example, abstract state $\$_3$ models all situations where all blocks are stacked, in which case the only action possible is to move the top block to the floor (A_{31}). In this case three abstract states generalize over 13 RMDP states. The abstraction levels are *exact aggregations* (i.e. partitions) of state-action spaces, and are closely related to *averagers* (Gordon, 1995). van Otterlo (2004) uses this in a *Q*-learning setting, and also in a *model-based* fashion (prioritized sweeping, Moore and Atkeson, 1993) where also a transition model between abstract states is learned. In essence, what is learned is the best abstract policy among all policies present in the representation, assuming that this *policy space*, can be obtained from a domain expert, or by other means.

Two closely related approaches are LOMDPs by Kersting and De Raedt (2004) and rQ-learning by Morales (2003). LOMDP abstraction levels are very similar to CARCASS, and they use *Q*-learning and a *logical TD*(λ)-algorithm to learn state values for abstract states. The rQ-framework is based on a *separate* definition of abstract states (*r-states*) and abstract actions (*r-actions*). The product space of r-states and r-actions induces a new (abstract) state-action space over which *Q*-learning can be performed. All approaches can be used to learn optimal policies in domains where prior knowledge exists. They can also be employed as part of other learning methods, e.g. to learn sub-policies in a given hierarchical policy. A related effort based on *Markov logic networks* was reported by Wang et al (2008b).

Initial investigations into automatically generating the abstractions have been reported (Song and Chen, 2007, 2008), and Morales (2004) employed *behavioral cloning* to learn *r-actions* from sub-optimal traces, generated by a human expert. AMBIL (Walker et al, 2007) too learns abstract state-action pairs from traces; given an abstraction level, it estimates an approximate model (as in CARCASS) and uses it to generate new abstract state-action pairs. Each value learning iteration a new representation is generated.

Table 8.2 Main model-free approaches with static generalization. Legend: BW=Blocks World, Cnj=conjunction of logical atoms, Q=Q-learning, PS=prioritized sweeping, TD=TD-learning, RF=relational features.

method	representation	algorithm	applications
CARCASS (van Otterlo, 2003, 2004)	Cnj + negation	Q, PS	BW, Tic-Tac-Toe
LOMDP (Kersting and De Raedt, 2004)	Cnj	Q, TD	BW
RQ (Morales, 2003) (Walker et al, 2004)	Cnj + negation RF	Q Q	Taxi, Chess, Grids Robocup simulation
rTD (Asgharbeygi et al, 2006)	RF	TD	Tic-Tac-Toe, Mini-Chess

A variation on fixed abstractions is to use abstract states as *relational features* for value function approximation. Each abstract state can be seen as a binary feature; whether a state is subsumed by it or not. Walker et al (2004) first generates semi-randomly a set of relational features, and learns the weights of a linear combination of the features to represent the value function for each action. A related technique is *relational temporal difference* learning (rTD), by Asgharbeygi et al (2006). It uses a set of concept definitions (similar to abstract states) that must be supplied by a designer and assigns a utility to each of them. The value function is then represented as a linear combination of the utilities for those concepts, weighted by the number of times a concept occurs. Table 8.2 summarizes this section.

8.4.2 Value Functions with Adaptive Generalization

For *adaptive* generalization, i.e. changing the representation during learning (PIAGET-3), ILP methods are usually used on state(-action) samples derived from interaction with the RMDP (the I(nduction) steps in Equation 8.4). In this section we will describe three types of methods: **i**) those that build logical abstractions (e.g. of value functions), **ii**) those based on distances and kernels and **iii**) probabilistic techniques.

Learning Value Function Abstractions

As said, the Q-RRL method (Džeroski et al, 1998) was the first approach towards model-free RL in RMDPs. It is a straightforward combination of *Q*-learning and ILP for generalization of *Q*-functions, and was tested on small deterministic Blocks Worlds. Q-RRL collects experience in the form of state-action pairs with corresponding *Q*-values. During an episode, actions are taken according to the current policy, based on the current *Q*-tree. After each episode a decision tree is induced

from the example set (see Fig. 8.3(**right**)). Q -trees can employ *background knowledge* (such as about the amount of and heights of towers, the number of blocks).

One problem with Q -functions however, is that they implicitly encode a *distance* to the goal, and they are dependent on the domain size in *families* of RMDPs. A Q -function represents more information than needed for selecting an optimal action. *P-learning* can be used to learn policies from the current Q -function and a training set (Džeroski et al, 2001). For each state s occurring in the training set, all possible actions in that state are evaluated and a P -value is computed as $P(s,a) = 1$ if $a = \arg \max_{a'} Q(s,a')$ and 0 otherwise. The P -tree represents the best policy relative to that Q -tree. In general, it will be less complex and generalize (better) over domains with different numbers of blocks. Independently, Lecoeuche (2001) showed similar results. Cole et al (2003) use a similar setup as Q-RRL, but upgrade the representation language to *higher-order logic* (HOL) (Lloyd, 2003).

An *incremental* extension to Q-RRL is the TG-algorithm (Driessens et al, 2001). It can be seen as a relational extension of the G -algorithm (Chapman and Kaelbling, 1991), and it *incrementally* builds Q -trees. Each leaf in a tree is now augmented with statistics about Q -values and the number of positive matches of examples. A node is split when it has seen enough examples and a test on the node's statistics becomes significant with high confidence. This mechanism removes the need for storing, retrieving and updating individual examples. Generating new tests is much more complex than in the propositional case, because the amount of possible splits is essentially unlimited and the number of possibilities grows further down the tree with the number of variables introduced in earlier nodes.

In a subsequent upgrade TGR of TG, Ramon et al (2007) tackle the problem of the *irreversibility* of the splits by adding a *tree restructuring* operation. This includes leaf or subtree pruning, and internal node revision. To carry out these operations statistics are now stored in *all* nodes in the tree. Special care is to be taken with variables in the tree when building and restructuring the tree. Another method that has used restructuring operations from the start is the *relational* UTree (rUTree) algorithm by Dabney and McGovern (2007). rUTree is a relational extension of the UTree algorithm by McCallum (1995). Because rUTree is instance-based, tests can be regenerated when needed for a split such that statistics do not have to be kept for all nodes, as in TGR. Another interesting aspect of rUTree is that it uses *stochastic sampling* (similar to the approach by Walker et al (2004), to cope with the large number of possible tests when splitting a node. Combining these last two aspects shows an interesting distinction with TG (and TGR). Whereas TG must keep all statistics and consider all tests, rUTree considers only a limited, sampled set of possible tests. In return, rUTree must often recompute statistics.

Finally, *first-order* XCS (FOXCS) by Mellor (2008) is a *learning classifier system* (e.g. see Lanzi, 2002) with relational rules. FOXCS' policy representation is similar to that of CARCASS, but each rule is augmented with an *accuracy* and each time an action is required, *all* rules that cover the current state-action pair considered, are taken into account. The accuracy can be used for both action

selection, and adaptation (and creation) of rules by an *evolutionary learning* algorithm. Experimental results on Blocks World instances show that FOXCS can compete with other approaches such as TG.

Generalization using Distances and Kernels

Instead of building logical abstractions several methods use other means for generalization over states modeled as relational interpretations.

The *relational instance based regression* method (RIB) by Driessens and Ramon (2003) uses *instance-based learning* (Aha et al, 1991) on ground relational states. The Q -function is represented by a set of well-chosen experienced examples. To look-up the value of a newly encountered state-action pair, a *distance* is computed between this pair and the stored pairs, and the Q -value of the new pair is computed as an average of the Q -values of pairs that it resembles. Special care is needed to maintain the right set of examples, by throwing away, updating and adding examples. Instance-based regression for Q -learning has been employed for propositional representations before but the challenge in relational domains is defining a suitable *distance* between two interpretations. For RIB, a *domain-specific* distance has to be defined beforehand. For example, in Blocks World problems, the distance between two states is computed by first renaming variables, by comparing the stacks of blocks in the state and finally by the *edit distance* (e.g. how many actions are needed to get from one state to another). Other background knowledge or declarative bias is not used, as the representation consists solely of ground states and actions. García-Durán et al (2008) used a more general instance-based approach in a policy-based algorithm. Katz et al (2008) use a relational instance-based algorithm in a robotic manipulation task. Their similarity measure is defined in terms of isomorphic subgraphs induced by the relational representation.

Later, the methods TG and RIB were combined by Driessens and Džeroski (2005), making use of the strong points of both methods. TG builds an explicit, structural model of the value function and – in practice – can only build up a coarse approximation. RIB is not dependent on a language bias and the instance-based nature is better suited for regression, but it does suffer from large numbers of examples that have to be processed. The combined algorithm – TRENDI builds up a tree like TG but uses an instance-based representation in the leaves of the tree. Because of this, new splitting criteria are needed and both the (language) bias for TG and RIB are needed for TRENDI. However, on deterministic Blocks World examples the new algorithm performs better on some aspects (such as computation time) than its parent techniques. Note that the rUTree algorithm is also a combination of an instance-based representation combined with a logical abstraction level in the form of a tree. No comparison has been reported yet. Rodrigues et al (2008) recently investigated the online behavior of RIB and their results indicate that the number of instances in the system is decreased when per-sample updates are used.

Compared to RIB, Gärtner et al (2003) take a more principled approach in the KBR algorithm to distances between relational states and use *graph kernels* and *Gaussian processes* for value function approximation in relational RL. Each

state-action pair is represented as a *graph* and a product kernel is defined for this class of graphs. The kernel is wrapped into a Gaussian radial basis function, which can be tuned to regulate the amount of generalization.

Probabilistic Approaches

Two additional, structurally adaptive, algorithms learn and use probabilistic information about the environment to optimize behavior, yet for different purposes.

SVRRL (Sanner, 2005) targets undiscounted, finite-horizon domains in which there is a single terminal reward. This enables viewing the value function as a *probability of success*, such that it can be represented as a *relational naive Bayes network*. The structure and the parameters of this network are learned simultaneously. The parameters can be computed using standard techniques based on the maximum likelihood. Two structure learning approaches are described for SVRRL and in both relational features (ground relational atoms) can be combined into joint features if they are more informative than the independent features' estimates. An extension DM-SVRRL uses *datamining* to focus structure learning on only those parts of the state space that are frequently visited (Sanner, 2006) finds frequently co-occurring features and turns them into joint features, which can later be used to build even larger features.

SVRRL is not based on TD-learning, but on probabilistic reasoning. MARLIE (Croonenborghs et al, 2007b) too uses probabilistic techniques, but employs it for transition model learning. It is one of the very few relational *model-based* RL algorithms (next to CARCASS). It learns how ground relational atoms change from one state to another. For each a *probability tree* is learned incrementally using a modified TG algorithm. Such trees represent for each ground instance of a predicate the probability that it will be true in the next state, given the current state and action. Using the model amounts to look ahead some steps in the future using an existing technique called *sparse sampling*. The original TG algorithm is used to store the *Q*-value function.

Table 8.3 summarizes some of the main characteristics. Detailed (experimental) comparison between the methods is still a task to be accomplished. A crucial aspect of the methods is the combination of representation and behavior learning. Fixed abstraction levels can provide convergence guarantees but are not flexible. Incremental algorithms such as TG and restructuring approaches such as uTree and TGR provide increasingly flexible function approximators, at the cost of increased computational complexity and extensive bookkeeping.

An important aspect in all approaches is the *representation* used for examples, and how examples are *generalized into abstractions*. Those based on logical abstractions have a number of significant advantages: **i)** they are more easy to generalize over problems of different domain size through the use of variables and **ii)** abstractions are usually more *comprehensible* and *transferrable*. On the other hand, logical abstractions are less suitable for finegrained regression. Methods such as TG have severe difficulties with some very simple Blocks World problems. Highly relational problems such as the Blocks World require complex patterns in their value functions

Table 8.3 Main model-free approaches with adaptive generalization that were discussed. Legend: BW=Blocks World, IB=instance based, NB=naive Bayes, HOL=higher-order logic.

method	representation	algorithm	applications
Q-RRL (Džeroski et al, 1998) (Lecoeuche, 2001) (Cole et al, 2003)	rel. tree rel. decision list HOL	Q-learning Q-learning Q-learning	BW dialogues BW
TG (Driessens et al, 2001)	incremental tree	Q-learning	BW
TGR (Ramon et al, 2007)	adaptive tree	Q-learning	BW
rUTree (Dabney and McGovern, 2007).	adaptive tree	Q-learning	BW, Tsume-Go
RIB (Driessens and Ramon, 2003)	IB	Q-learning	BW
TRENDI (Driessens and Džeroski, 2005)	tree + IB	Q-learning	BW
KBR (Gärtner et al, 2003)	kernels	Q-learning	BW
MARLIE (Croonenborghs et al, 2007b)	prob. rules	model-based	BW
SVRRL (Sanner, 2005)	relational NB	Bayesian	Backgammon

and learning these in a typical RL learning process is difficult. Other methods that base their estimates (in part) on instance-based representations, kernels or first-order features are more suitable because they can provide more smooth approximations of the value function.

8.4.3 Policy-Based Solution Techniques

Policy-based approaches have a simple structure: one starts with a policy structure $\tilde{\Pi}^0$, generates samples (**S**) by interaction with the underlying *RMDP*, generates (**D**) a new abstract policy $\tilde{\Pi}^1$, and so on. The general structure is the following:

$$\tilde{\Pi}^0 \xrightarrow{S} \{\langle s, a, q \rangle\} \xrightarrow{I} \tilde{\Pi}^1 \xrightarrow{S} \{\langle s, a, q \rangle\} \xrightarrow{I} \tilde{\Pi}^2 \longrightarrow \dots$$

The representation used is usually the decision-rule-like structure we have presented before. An important difference with value learning is that no explicit representations of \tilde{Q} are required. At each iteration of these *approximate policy iteration* algorithms, the current policy is used to gather useful learning experience – which can be samples of state-action pairs, or the amount of reward gathered by that policy – which is then used to generate a new policy structure.

Evolutionary Relational Policy Search

The first type of policy-based approaches are *evolutionary* approaches, which have been used in propositional RL before (Moriarty et al, 1999). Distinct features of these approaches are that they usually maintain a *population* (i.e. a set) of policy structures, and that they assign a single-valued *fitness* to each policy (or policy rule) based on how it performs on the problem. The fitness is used to combine or modify policies, thereby searching directly in *policy space*.

The Grey system (Muller and van Otterlo, 2005) employs simple relational decision list policies to evolve Blocks World policies. GAPI (van Otterlo and De Vuyst, 2009) is a similar approach based on genetic algorithms, and evolves *probabilistic* relational policies. Gearhart (2003) employs a related technique (*genetic programming* (GP)) in the real-time strategy FreeCraft domain used by Guestrin et al (2003a). Results show that it compares well to Guestrin et al's approach, though it has difficulties with rarely occurring states. Castilho et al (2004) focus on STRIPS planning problems, but unlike Grey for example, each chromosome encodes a full plan, meaning that the approach searches in *plan space*. Both Kochenderfer (2003) and Levine and Humphreys (2003) do search in policy space, and both use a GP algorithm. Levine and Humphreys learn decision list policies from optimal plans generated by a planner, which are then used in a *policy restricted* planner. Kochenderfer allows for *hierarchical* structure in the policies, by simultaneously evolving sub-policies that can call each other. Finally, Baum (1999) describes the Hayek machines that use evolutionary methods to learn policies for Blocks Worlds. An additional approach (FOXCS) was discussed in a previous section.

Policy Search as Classification

A second type of approach uses ILP algorithms to learn the structure of the policy from sampled state-action pairs. This essentially transforms the RL process into a sequence of supervised (classification) learning tasks in which an abstract policy is repeatedly induced from a (biased) set of state-action pairs sampled using the previous policy structure. The challenge is to get good samples either by getting them from optimal traces (e.g. generated by a human expert, or a planning algorithm), or by smart trajectory sampling from the current policy. Both types of approaches are PIAGET-3 , combining structure and parameter learning in a single algorithm.

The model-based approach by Yoon et al (2002) induces policies from optimal traces generated by a planner. The algorithm can be viewed upon as an extension of the work by Martin and Geffner (2000) and Khordon (1999) to stochastic domains. Khordon (1999) studied the induction of deterministic policies for undiscounted, goal-based planning domains, and proved general PAC-bounds on the number of samples needed to obtain policies of a certain quality.

The LRW-API approach by Fern et al (2006) unifies, and extends, the aforementioned approaches into one practical algorithm. LRW-API is based on a concept language (taxonomic syntax), similar to Martin and Geffner (2000)'s approach, and targeted at complex, probabilistic planning domains, as is Yoon et al (2002)'s approach. LWR-API shares its main idea of iteratively inducing policy structures (i.e. *approximate policy iteration*, API) and using the current policy to bias the generation of samples to induce an improved policy. Two main improvements of LRW-API relative to earlier approaches lie in the sampling process of examples, and in the bootstrapping process. Concerning the first, LRW-API uses *policy rollout* (Boyan and Moore, 1995) to sample the current policy. That is, it estimates all action

values for the current policy for a state s by drawing w trajectories of length h , where each trajectory is the result of starting in state s , doing a , and following the policy for $h - 1$ more steps. Note that this requires a simulator that can be sampled from any state, at any moment in time. The *sampling width* w and *horizon* h are parameters that trade-off variance and computation time. A second main improvement of LRW-API is the bootstrapping process, which amounts here to *learning from random worlds* (LRW). The idea is to learn complex problems by first starting on simple problems and then iteratively solving more and more complex problem instances. Each problem instance is generated by a *random walk of length n* through the underlying RMDP, and by increasing n problems become more complex.

Policy Gradient Approaches

The last policy-based technique is based on *policy-gradient* approaches (Sutton et al, 2000). For this, one needs a *parameterized* relational policy to use the gradient of the policy parameters to optimize the policy.

Itoh and Nakamura (2004) describe a first approach for partially observable RMDPs in which policies are represented as a relational decision list. The hand-coded policies make use of a memory consisting of a limited number of memory bits. The algorithm is tested in a maze-like domain where planning is sometimes useful and the problem is to learn *when* it is useful. This is done by treating the policy as stochastic where the probabilities for the policy rules are used for exploration and learned via gradient descent techniques. Gretton (2007a) developed ROPG which learns *temporally extended* policies for domains with *non-Markovian* rewards. Policy gradients are used naturally, and two ways of generating policy rules are used: **i**) a sampling approach, and **ii**) computing rules from the problem specification. Despite slow convergence in general, the RPOG approach learned useful general policies in an elevator scheduling problem. Recently, in the *non-parametric policy gradient* (NPPG) approach, Kersting and Driessens (2008) applied the idea of *gradient boosting*. NPPG builds ensembles of regression models for the value function, thereby expanding the representation while learning (PIAGET-3). Since it just *lifts* the level at which gradients are computed, it works for propositional and relational domains in a similar way.

8.5 Models, Hierarchies, and Bias

The previous two parts of this chapter have surveyed relational upgrades of traditional model-free and model-based algorithms. In this section we take a look at relational techniques used for other aspects of the RMDP framework. We distinguish three groups: **i**) those that learn (probabilistic) models of the RMDP, **ii**) those that impose structure on policies, and **iii**) those that generally bias the learner.

Table 8.4 Main model-free policy-based approaches that were discussed. Legend: BW=Blocks World, PG=policy gradient, Q=Q-learning, PS=prioritized sweeping, TD=TD-learning, DL=decision list, GD=gradient descent.

method	representation	algorithm	applications
Grey (Muller and van Otterlo, 2005)	DL	evo	BW
FOX-CS (Mellor, 2008)	rules	Q-evo	BW
(Gearhart, 2003)	PRM	GP	FreeCraft
(Yoon et al, 2002)	DL	planning	BW
LRW-API Fern et al (2007)	DL	rollout	planning
(Itoh and Nakamura, 2004)	prob. rules	PG/GD	maze
RPOG (Gretton, 2007a)	rules	PG	elevator planning
NPPG (Kersting and Driessens, 2008)	regression trees	PG	BW
GAPI (van Otterlo and De Vuyst, 2009)	prob. DL	evolution	BW, Goldfinder

Learning Models of the World

Learning world models is one of the most useful things an agent can do. Transition models embody *knowledge* about the environment that can be exploited in various ways. Such models can be used for more efficient RL algorithms, for model-based DP algorithms, and furthermore, they can often be transferred to other, similar environments. There are several approaches that learn general operator models from interaction. All model-based approaches in Section 8.3 on the contrary, take for granted that a complete, logical model is available.

Usually, *model learning* amounts to learning general operator descriptions, such as the STRIPS rules earlier in this text. However, simpler models can already be very useful and we have seen examples such as the partial models used in MARLIE and the abstract models learned for CARCASS. Another example is the more specialized action model learning employed by Morales (2004) who learns *r*-actions using behavioral cloning. A recent approach by Halbritter and Geibel (2007) is based on graph kernels, which can be used to store transition models without the use of logical abstractions such as used in most action formalisms. A related approach in robotics by Mourão et al (2008) is based on *kernel perceptrons* but is restricted to learning (deterministic) *effects* of STRIPS-like actions.

Learning aspects of (STRIPS) operators from (planning) data is an old problem (e.g. see Vere, 1977) and several older works give partial solutions (e.g. only learning effects of actions). However, learning *full* models, with the added difficulty of *probabilistic* outcomes, is complex since it involves learning both logical structures and parameters from data. Remember the probabilistic STRIPS action we have defined earlier. This *move* action has two different, probabilistic outcomes. Learning such a description from data involves a number of aspects. First, the logical descriptions of the pre- and post-conditions have to be learned from data. Second, the learning algorithm has to infer how many outcomes an action has. Third, probabilities must be estimated for each outcome of the action. For the relational case, early approaches by Gil (1994) and Wang (1995) learn *deterministic* operator

descriptions from data, by interaction with simulated worlds. More recent work also targets *incomplete state information* (Wu et al, 2005; Zhuo et al, 2007), or considers *sample complexity* aspects (Walsh and Littman, 2008).

For general RMDPs though, we need *probabilistic* models. In the propositional setting, the earliest learning approach was described by Oates and Cohen (1996). For the relational models in the context of RMDPs the first approach was reported by Pasula et al (2004), using a three-step greedy search approach. First, a search is performed through the set of rule sets using standard ILP operators. Second, it finds the best set of outcomes, given a context and an action. Third, it learns a probability distribution over sets of outcomes. The learning process is *supervised* as it requires a dataset of state-action-state pairs taken from the domain. As a consequence, the rules are only valid on this set, and care has to be taken that it is representative for the domain. Experiments on Blocks Worlds and logistics domains show the robustness of the approach. The approach was later extended by Zettlemoyer et al (2005) who added *noise outcomes*, i.e. outcomes which are difficult to model exactly, but which do happen (like knocking over a blocks tower and scattering all blocks on the floor). Another approach was introduced by Safaei and Ghassem-Sani (2007), which works *incrementally* and combines planning and learning.

Hierarchies and Agents

Hierarchical approaches can be naturally incorporated into relational RL, yet not many techniques have been reported so far, although some cognitive architectures (and *sapient* agents, cf. van Otterlo et al, 2007)) share these aspects. An advantage of *relational* HRL is that parameterizations of sub-policies and goals naturally arise, through logical variables. For example, a Blocks World task such as $\text{on}(X,Y)$, where X and Y can be instantiated using any two blocks, can be *decomposed* into two tasks. First, all blocks must be removed from X and Y and then X and Y should be moved on top of each other. Note that the first task also consists of two subtasks, supporting even further decomposition. Now, by first learning policies for each of these subtasks, the individual learning problems for each of these subtasks are much simpler and the resulting skills might be reused. Furthermore, learning such subtasks can be done by any of the model-free algorithms in Section 8.4. Depending on the representation that is used, policies can be structured into hierarchies, facilitating learning in more complex problems.

Simple forms of *options* are straightforward to model relationally (see Croonenborghs et al, 2007a, for initial ideas). Driessens and Blockeel (2001) presented an approach based on the Q-RRL algorithm to learn two goals *simultaneously*, which can be considered a simple form of hierarchical decomposition. Aycenina (2002) uses the original Q-RRL system to build a complete system. A number of subgoals is given and separate policies are learned to achieve them. When learning a more complex task, instantiated sub-policies can be used as new actions. A related system by Roncagliolo and Tadepalli (2004) uses batch learning on a set of examples to learn values for a given relational hierarchy. Andersen (2005) presents the most thorough investigation using MAXQ hierarchies adapted to relational

representations. The work uses the Q-RRL framework to induce local Q -trees and P -trees, based on a manually constructed hierarchy of subgoals. A shortcoming of hierarchical relational systems is still that they assume the hierarchy is given beforehand. Two related systems combine planning and learning, which can be considered as generating hierarchical abstractions by planning, and using RL to learn concrete policies for behaviors. The approach by Ryan (2002) uses a planner to build a high-level task hierarchy, after which RL is used to learn the subpolicies. The method by Grounds and Kudenko (2005) is based on similar ideas. Also in cognitive architectures, not much work has been reported yet, with notable exceptions of model-learning in Icarus (Shapiro and Langley, 2002) and RL in SOAR (Nason and Laird, 2004).

Hierarchical learning can be seen as first decomposing a policy, and then do learning. *Multi-agent* (Wooldridge, 2002) decompositions are also possible. Letia and Precup (2001) report on multiple agents, modeled as *independent reinforcement learners* who do not communicate, but act in the same environment. Programs specify initial plans and knowledge about the environment, and complex actions induce semi-MDPs, and learning is performed by model-free RL methods based on *options*. A similar approach by Hernandez et al (2004) is based on the *belief-desires-intentions* model. Finzi and Lukasiewicz (2004a) introduce GTGolog, a *game-theoretic* language, which integrates explicit agent programming with game-theoretic multi-agent planning in Markov Games. Along this direction Finzi and Lukasiewicz (2004b) introduce *relational Markov Games* which can be used to abstract over multi-agent RMDPs and to compute *Nash policy pairs*.

Bias

Solution algorithms for RMDPs can be helped in many ways. A basic distinction is between helping the solution of the current problem (*bias*, *guidance*), and helping the solution of related problems (*transfer*).

Concerning the first, one idea is to supply a policy to the learner that can *guide* it towards useful areas in the state space (Driessens and Džeroski, 2002) by generating semi-optimal traces of behavior that can be used as experience by the learner. Related to guidance is the usage of *behavioral cloning* based on human generated traces, for example by (Morales, 2004) and Cocora et al (2006), the use of optimal plans by Yoon et al (2002), or the random walk approach by Fern et al (2006), which uses a guided exploration of *domain instantiations*. In the latter, first only easy instantiations are generated and the difficulty of the problems is increased in accordance with the current policy's quality. Domain sampling was also used by Guestrin et al (2003a). Another way to help the learner is by structuring the domain itself, for example by imposing a strong *topological structure* (Lane and Wilson, 2005). Since many of these guidance techniques are essentially representation-independent, there are many more existing (propositional) algorithms to be used for RMDPs.

A second way to help the learner, is by *transferring* knowledge from other tasks. *Transfer learning* – in a nutshell – is leveraging learned knowledge on a *source task* to improve learning on a related, but different *target task*. It is particularly

appropriate for learning agents that are meant to persist over time, changing flexibly among tasks and environments. Rather than having to learn each task from scratch, the goal is to take advantage of its past experience to speed up learning (see Stone, 2007). Transfer is much representation-dependent, and the use of (declarative) FOL formalisms in relational RL offers good opportunities.

In the more restricted setting of (relational) RL there are several possibilities. For example, the source and target problems can differ in the goal that must be reached, but possibly the transition model and reward model can be transferred. Sometimes a specific state abstraction can be transferred between problems (e.g. Walsh et al, 2006). Or, possibly some knowledge about actions can be transferred, although they can have slightly different effects in the source and target tasks. Sometimes a complete policy can be transferred, for example a *stacking* policy for a Blocks World can be transferred between worlds of varying size. Another possibility is to transfer solutions to *subproblems*, e.g. a sub-policy. Very often in relational domains, transfer is possible by the intrinsic nature of relational representations alone. That is, in Blocks Worlds, in many cases a policy that is learned for a task with n blocks will work for $m > n$ blocks.

So far, a few approaches have approached slightly more general notions of transfer in relational RL. Several approaches are based on transfer of learned skills in hierarchical decompositions, e.g. (Croonenborghs et al, 2007a) and (Torrey et al, 2007), and an extension of the latter using Markov logic networks (Torrey et al, 2008). Explicit investigations into transfer learning were based on methods we have discussed such as in rTD (Stracuzzi and Asgharbeigi, 2006) and the work by García-Durán et al (2008) on transferring instance-based policies in deterministic planning domains.

8.6 Current Developments

In this chapter so far, we have discussed an important selection of the methods described in (van Otterlo, 2009b). In this last section we briefly discuss some recent approaches and trends in solution techniques for RMDPs that appeared very recently. These topics also mark at least two areas with lots of open problems and potential; general logical-probabilistic engines, and partially observable problems.

Probabilistic Inference Frameworks

In many solution techniques we have discussed, RMDPs are tackled by upgrading MDP-specific representations, solution algorithms and methodology to the relational domain. Another route that has caught considerable interest nowadays comes from seeing decision-theoretic problems as problems that can be solved by general probabilistic and decision-theoretic reasoning engines. This direction starts with first viewing *planning* as a specific instance of *probabilistic inference* (Toussaint, 2009). For any MDP one can find a formulation where basically the task is to find that policy π that will have the *highest probability* of reaching the goal. This connection

between planning and probabilistic inference opens up many possibilities to setup a probabilistic planning problem as a general probabilistic inference problem and use optimized strategies for finding optimal plans.

Lang and Toussaint (2009) directly implement these ideas in the context of learned rules based on the technique by Pasula et al (2004). The relational probabilistic transition model is then transformed into a Bayesian network and standard inference techniques are used for (re-)planning. Later it was extended by incorporating both *forward* and *backward* planning Lang and Toussaint (2010). The transformation into propositional Bayesian networks makes it also easy to connect to lower-level probabilistic reasoning patterns, resulting in an integrated framework for high-level relational learning and planning with low-level behaviors on a real robotic arm (Toussaint et al, 2010). A downside of translating to Bayesian networks though, is that these networks typically blow up in size, and planning is still essentially done on a propositional level. A more logically-oriented approach was described by Thon et al (2009) who used SRL to learn probabilistic action models which can be directly translated into standard relational planning languages.

A more generic solution which is developing currently, is to extend SRL systems into full probabilistic *programming languages* with decision-theoretic capabilities. Many ideas were developed early on, mainly in the *independent choice logic* (Poole, 1997), and recently approaches appear that employ state-of-the-art reasoning and/or learning techniques. The DT-ProbLog language is the first probabilistic decision-theoretic programming capable of efficiently computing optimal (and approximate) plans (Van den Broeck et al, 2010). It is based on the ProbLog language for probabilistic reasoning, but it extends it with *reward facts*. Based on a set of *decision facts*, DT-ProbLog computes possible plans and stores their values compactly in algebraic decision diagrams (ADD). Optimization of the ADD gives the optimal (or approximated) set of decisions. So far, the language deals efficiently with *simultaneous* actions, i.e. multiple decisions have to be made in a problem, but the sequential aspect is not yet taken into account (such problems can be modeled and solved, but solving them efficiently is a step to be taken). It successfully makes *direct marketing* decisions for a huge social network. A related approach based on Markov logic networks (Nath and Domingos, 2009) can model similar problems, but only supports approximate reasoning. Another approach by Chen and Muggleton (2010) is based on stochastic logic programming but lacks an efficient implementation. Yet another approach along the same direction is that by Saad (2008) which is based on *answer set programming*. The central hypothesis in all these recent approaches is that current progress in SRL systems will become available to solve decision-theoretic problems such as RMDPs too. A related direction of employing RL in programming languages (Simpkins et al, 2008) is expected to become important in that respect as well.

Relational POMDPs

Virtually all methods presented so far assume fully-observable first-order MDPs. Extensions towards more complex models beyond this Markov assumption are

almost unexplored so far, and in (van Otterlo, 2009b) we mentioned some approaches that have went up this road.

Wingate et al (2007) present the first steps towards relational KR in *predictive representations of states* (Littman et al, 2001). Although the representation is still essentially propositional, they do capture some of Blocks World structure in a much richer framework than MDPs. Zhao and Doshi (2007) introduce a semi-Markov extension to the situation calculus approach in SDP in the Haley system for *web-based services*. Although no algorithm for solving the induced first-order SMDP is given, the approach clearly shows that the formalization can capture useful temporal structures. Gretton (2007a,b) compute *temporally extended policies* for domains with non-Markovian rewards, using a *policy gradient* approach and we have discussed it in the previous chapter.

The first contribution to the solution of *first-order* POMDPs is given by Wang (2007). Although modeling POMDPs using FOL formalisms has been done before (e.g. see Geffner and Bonet, 1998; Wang and Schmolze, 2005), Wang is the first to upgrade an existing POMDP solution algorithm to the first-order case. It takes the FODD formalism we have discussed earlier and extends it to model *observations*, conditioned on states. Based on the clear connections between regression-based backups in IDP algorithms and value backups over belief states, Wang upgrades the *incremental pruning* algorithm (see Kaelbling et al, 1998) to the first-order case.

Recently some additional efforts into relational POMDPs have been described, for example by Lison (2010) in dialogue systems, and by both Wang and Khordon (2010) and Sanner and Kersting (2010) who describe basic algorithms for such POMDPs along the lines of IDP.

New Methods and Applications

In addition to inference engines and POMDPs, other methods have appeared recently. In the previous we have already mentioned the evolutionary technique GAPI (van Otterlo and De Vuyst, 2009), and furthermore Neruda and Slusny (2009) provide a performance comparison between relational RL and evolutionary techniques.

New techniques for learning models include *instance based* techniques for deterministic action models in the SOAR cognitive architecture (Xu and Laird, 2010), and *incremental* learning of action models in the context of noise by Rodrigues et al (2010). Both represent new steps towards learning transition models that can be used for planning and FODTR. A related technique by Vargas-Govea and Morales (2009) learns *grammars* for sequences of actions, based on sequences of low-level sensor readings in a robotic context.

In fact, domains such as robotics are interesting application areas for relational RL. Both Vargas and Morales (2008) and Hernández and Morales (2010) apply relational techniques for navigation purposes in a robotic domain. The first approach learns *teleo-reactive programs* from traces using behavioral cloning. The second combines relational RL and continuous actions. Another interesting area for relational RL is *computer vision*, and initial work in this direction is reported by Hming and Peters (2009) who apply it for object recognition.

Just like in (van Otterlo, 2009b) we also pay attention to recent PhD theses that were written on topics in relational RL. Five more theses have appeared, on model-assisted approaches by Croonenborghs (2009), on transfer learning by Torrey (2009), on object-oriented representations in RL by Diuk (2010), on efficient model learning by Walsh (2010) and on FODTR by Joshi (2010).

8.7 Conclusions and Outlook

In this chapter we have surveyed the field of relational RL, summarizing the main techniques discussed in (van Otterlo, 2009b). We have discussed large subfields such as model-based and model-free algorithms, and in addition hierarchies, models, POMDPs and much more.

There are many future directions for relational RL. For one part, it can be noticed that not many techniques make use of efficient data structures. Some model-based techniques use binary (or algebraic) data structures, but many other methods may benefit since relational RL is a computationally complex task. The use of more efficient inference engines has started with the development of languages such as DT-Problog, but it is expected that the next few years more efforts along these lines will be developed in the field of SRL. In the same direction, POMDP techniques and probabilistic model learning can be explored too from the viewpoint of SRL.

In terms of application areas, robotics, computer vision, and web and social networks may yield interesting problems. The connection to *manipulation* and *navigation* in robotics is easily made and not many relational RL techniques have been used so far. Especially the grounding and anchoring of sensor data to relational representations is a hard problem and largely unsolved.

References

- Aha, D., Kibler, D., Albert, M.: Instance-based learning algorithms. *Machine Learning* 6(1), 37–66 (1991)
- Alpaydin, E.: *Introduction to Machine Learning*. The MIT Press, Cambridge (2004)
- Andersen, C.C.S.: Hierarchical relational reinforcement learning. Master's thesis, Aalborg University, Denmark (2005)
- Asgharbeygi, N., Stracuzzi, D.J., Langley, P.: Relational temporal difference learning. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 49–56 (2006)
- Aycenina, M.: Hierarchical relational reinforcement learning. In: Stanford Doctoral Symposium (2002) (unpublished)
- Baum, E.B.: Toward a model of intelligence as an economy of agents. *Machine Learning* 35(2), 155–185 (1999)
- Baum, E.B.: *What is Thought?* The MIT Press, Cambridge (2004)
- Bergadano, F., Gunetti, D.: *Inductive Logic Programming: From Machine Learning to Software Engineering*. The MIT Press, Cambridge (1995)

- Bertsekas, D.P., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific, Belmont (1996)
- Boutilier, C., Poole, D.: Computing optimal policies for partially observable markov decision processes using compact representations. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 1168–1175 (1996)
- Boutilier, C., Dean, T., Hanks, S.: Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11, 1–94 (1999)
- Boutilier, C., Dearden, R.W., Goldszmidt, M.: Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121(1-2), 49–107 (2000)
- Boutilier, C., Reiter, R., Price, B.: Symbolic dynamic programming for first-order MDP's. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 690–697 (2001)
- Boyan, J.A., Moore, A.W.: Generalization in reinforcement learning: Safely approximating the value function. In: Proceedings of the Neural Information Processing Conference (NIPS), pp. 369–376 (1995)
- Brachman, R.J., Levesque, H.J.: Knowledge Representation and Reasoning. Morgan Kaufmann Publishers, San Francisco (2004)
- Castilho, M.A., Kunzle, L.A., Lecheta, E., Palodeto, V., Silva, F.: An Investigation on Genetic Algorithms for Generic STRIPS Planning. In: Lemaître, C., Reyes, C.A., González, J.A. (eds.) IBERAMIA 2004. LNCS (LNAI), vol. 3315, pp. 185–194. Springer, Heidelberg (2004)
- Chapman, D., Kaelbling, L.P.: Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 726–731 (1991)
- Chen, J., Muggleton, S.: Decision-theoretic logic programs. In: Proceedings of the International Conference on Inductive Logic Programming (ILP) (2010)
- Cocora, A., Kersting, K., Plagemann, C., Burgard, W., De Raedt, L.: Learning relational navigation policies. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2006)
- Cole, J., Lloyd, J.W., Ng, K.S.: Symbolic learning for adaptive agents. In: Proceedings of the Annual Partner Conference, Smart Internet Technology Cooperative Research Centre (2003), http://csl.anu.edu.au/jwl/crc_paper.pdf
- Croonenborghs, T.: Model-assisted approaches for relational reinforcement learning. PhD thesis, Department of Computer Science, Catholic University of Leuven, Belgium (2009)
- Croonenborghs, T., Driessens, K., Bruynooghe, M.: Learning relational options for inductive transfer in relational reinforcement learning. In: Proceedings of the International Conference on Inductive Logic Programming (ILP) (2007a)
- Croonenborghs, T., Ramon, J., Blockeel, H., Bruynooghe, M.: Online learning and exploiting relational models in reinforcement learning. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 726–731 (2007b)
- Dabney, W., McGovern, A.: Utile distinctions for relational reinforcement learning. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 738–743 (2007)
- de la Rosa, T., Jimenez, S., Borrajo, D.: Learning relational decision trees for guiding heuristic planning. In: Proceedings of the International Conference on Artificial Intelligence Planning Systems (ICAPS) (2008)
- De Raedt, L.: Logical and Relational Learning. Springer, Heidelberg (2008)
- Dietterich, T.G., Flann, N.S.: Explanation-based learning and reinforcement learning: A unified view. *Machine Learning* 28(503), 169–210 (1997)

- Diuk, C.: An object-oriented representation for efficient reinforcement learning. PhD thesis, Rutgers University, Computer Science Department (2010)
- Diuk, C., Cohen, A., Littman, M.L.: An object-oriented representation for efficient reinforcement learning. In: Proceedings of the International Conference on Machine Learning (ICML) (2008)
- Driessens, K., Blockeel, H.: Learning Digger using hierarchical reinforcement learning for concurrent goals. In: Proceedings of the European Workshop on Reinforcement Learning, EWRL (2001)
- Driessens, K., Džeroski, S.: Integrating experimentation and guidance in relational reinforcement learning. In: Proceedings of the Nineteenth International Conference on Machine Learning, pp. 115–122 (2002)
- Driessens, K., Džeroski, S.: Combining model-based and instance-based learning for first order regression. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 193–200 (2005)
- Driessens, K., Ramon, J.: Relational instance based regression for relational reinforcement learning. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 123–130 (2003)
- Driessens, K., Ramon, J., Blockeel, H.: Speeding Up Relational Reinforcement Learning Through the Use of an Incremental First Order Decision Tree Learner. In: Flach, P.A., De Raedt, L. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, pp. 97–108. Springer, Heidelberg (2001)
- Džeroski, S., De Raedt, L., Blockeel, H.: Relational reinforcement learning. In: Shavlik, J. (ed.) Proceedings of the International Conference on Machine Learning (ICML), pp. 136–143 (1998)
- Džeroski, S., De Raedt, L., Driessens, K.: Relational reinforcement learning. Machine Learning 43, 7–52 (2001)
- Feng, Z., Dearden, R.W., Meuleau, N., Washington, R.: Dynamic programming for structured continuous Markov decision problems. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI), pp. 154–161 (2004)
- Fern, A., Yoon, S.W., Givan, R.: Approximate policy iteration with a policy language bias: Solving relational markov decision processes. Journal of Artificial Intelligence Research (JAIR) 25, 75–118 (2006); special issue on the International Planning Competition 2004
- Fern, A., Yoon, S.W., Givan, R.: Reinforcement learning in relational domains: A policy-language approach. The MIT Press, Cambridge (2007)
- Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2(2) (1971)
- Finney, S., Gardiol, N.H., Kaelbling, L.P., Oates, T.: The thing that we tried Didn't work very well: Deictic representations in reinforcement learning. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI), pp. 154–161 (2002)
- Finzi, A., Lukasiewicz, T.: Game-theoretic agent programming in Golog. In: Proceedings of the European Conference on Artificial Intelligence (ECAI) (2004a)
- Finzi, A., Lukasiewicz, T.: Relational Markov Games. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 320–333. Springer, Heidelberg (2004)
- García-Durán, R., Fernández, F., Borrajo, D.: Learning and transferring relational instance-based policies. In: Proceedings of the AAAI-2008 Workshop on Transfer Learning for Complex Tasks (2008)
- Gardiol, N.H., Kaelbling, L.P.: Envelope-based planning in relational MDPs. In: Proceedings of the Neural Information Processing Conference (NIPS) (2003)

- Gardiol, N.H., Kaelbling, L.P.: Adaptive envelope MDPs for relational equivalence-based planning. Tech. Rep. MIT-CSAIL-TR-2008-050, MIT CS & AI Lab, Cambridge, MA (2008)
- Gärtner, T., Driessens, K., Ramon, J.: Graph kernels and Gaussian processes for relational reinforcement learning. In: Proceedings of the International Conference on Inductive Logic Programming (ILP) (2003)
- Gearhart, C.: Genetic programming as policy search in Markov decision processes. In: Genetic Algorithms and Genetic Programming at Stanford, pp. 61–67 (2003)
- Geffner, H., Bonet, B.: High-level planning and control with incomplete information using pomdps. In: Proceedings Fall AAAI Symposium on Cognitive Robotics (1998)
- Gil, Y.: Learning by experimentation: Incremental refinement of incomplete planning domains. In: Proceedings of the International Conference on Machine Learning (ICML) (1994)
- Gordon, G.J.: Stable function approximation in dynamic programming. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 261–268 (1995)
- Gretton, C.: Gradient-based relational reinforcement-learning of temporally extended policies. In: Proceedings of the International Conference on Artificial Intelligence Planning Systems (ICAPS) (2007a)
- Gretton, C.: Gradient-based relational reinforcement learning of temporally extended policies. In: Workshop on Artificial Intelligence Planning and Learning at the International Conference on Automated Planning Systems (2007b)
- Gretton, C., Thiébaut, S.: Exploiting first-order regression in inductive policy selection. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI), pp. 217–225 (2004a)
- Gretton, C., Thiébaut, S.: Exploiting first-order regression in inductive policy selection (extended abstract). In: Proceedings of the Workshop on Relational Reinforcement Learning at ICML 2004 (2004b)
- Groote, J.F., Tveretina, O.: Binary decision diagrams for first-order predicate logic. *The Journal of Logic and Algebraic Programming* 57, 1–22 (2003)
- Grounds, M., Kudenko, D.: Combining Reinforcement Learning with Symbolic Planning. In: Tuyls, K., Nowe, A., Guessoum, Z., Kudenko, D. (eds.) ALAMAS 2005, ALAMAS 2006, and ALAMAS 2007. LNCS (LNAI), vol. 4865, pp. 75–86. Springer, Heidelberg (2008)
- Guestrin, C.: Planning under uncertainty in complex structured environments. PhD thesis, Computer Science Department, Stanford University (2003)
- Guestrin, C., Koller, D., Gearhart, C., Kanodia, N.: Generalizing plans to new environments in relational MDPs. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 1003–1010 (2003a)
- Guestrin, C., Koller, D., Parr, R., Venkataraman, S.: Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research (JAIR)* 19, 399–468 (2003b)
- Halbritter, F., Geibel, P.: Learning Models of Relational MDPs Using Graph Kernels. In: Gelbukh, A., Kuri Morales, Á.F. (eds.) MICAI 2007. LNCS (LNAI), vol. 4827, pp. 409–419. Springer, Heidelberg (2007)
- Hanks, S., McDermott, D.V.: Modeling a dynamic and uncertain world I: Symbolic and probabilistic reasoning about change. *Artificial Intelligence* 66(1), 1–55 (1994)
- Guerra-Hernández, A., Fallah-Seghrouchni, A.E., Soldano, H.: Learning in BDI Multi-Agent Systems. In: Dix, J., Leite, J. (eds.) CLIMA 2004. LNCS (LNAI), vol. 3259, pp. 218–233. Springer, Heidelberg (2004)

- Hernández, J., Morales, E.F.: Relational reinforcement learning with continuous actions by combining behavioral cloning and locally weighted regression. *Journal of Intelligent Systems and Applications* 2, 69–79 (2010)
- Häming, K., Peters, G.: Relational Reinforcement Learning Applied to Appearance-Based Object Recognition. In: Palmer-Brown, D., Draganova, C., Pimenidis, E., Mouratidis, H. (eds.) *EANN 2009. Communications in Computer and Information Science*, vol. 43, pp. 301–312. Springer, Heidelberg (2009)
- Hölldobler, S., Skvortsova, O.: A logic-based approach to dynamic programming. In: *Proceedings of the AAAI Workshop on Learning and Planning in Markov Processes - Advances and Challenges* (2004)
- Itoh, H., Nakamura, K.: Towards learning to learn and plan by relational reinforcement learning. In: *Proceedings of the ICML Workshop on Relational Reinforcement Learning* (2004)
- Joshi, S.: First-order decision diagrams for decision-theoretic planning. PhD thesis, Tufts University, Computer Science Department (2010)
- Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 99–134 (1998)
- Kaelbling, L.P., Oates, T., Gardiol, N.H., Finney, S.: Learning in worlds with objects. In: *The AAAI Spring Symposium* (2001)
- Karabaev, E., Skvortsova, O.: A heuristic search algorithm for solving first-order MDPs. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)* (2005)
- Karabaev, E., Rammé, G., Skvortsova, O.: Efficient symbolic reasoning for first-order MDPs. In: *ECAI Workshop on Planning, Learning and Monitoring with Uncertainty and Dynamic Worlds* (2006)
- Katz, D., Pyuro, Y., Brock, O.: Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. In: *Proceedings of Robotics: Science and Systems IV* (2008)
- Kersting, K., De Raedt, L.: Logical Markov decision programs and the convergence of $\text{TD}(\lambda)$. In: *Proceedings of the International Conference on Inductive Logic Programming (ILP)* (2004)
- Kersting, K., Driessens, K.: Non-parametric gradients: A unified treatment of propositional and relational domains. In: *Proceedings of the International Conference on Machine Learning (ICML)* (2008)
- Kersting, K., van Otterlo, M., De Raedt, L.: Bellman goes relational. In: *Proceedings of the International Conference on Machine Learning (ICML)* (2004)
- Kharden, R.: Learning to take actions. *Machine Learning* 35(1), 57–90 (1999)
- Kochenderfer, M.J.: Evolving Hierarchical and Recursive Teleo-Reactive Programs Through Genetic Programming. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E.P.K., Poli, R., Costa, E. (eds.) *EuroGP 2003. LNCS*, vol. 2610, pp. 83–92. Springer, Heidelberg (2003)
- Lane, T., Wilson, A.: Toward a topological theory of relational reinforcement learning for navigation tasks. In: *Proceedings of the International Florida Artificial Intelligence Research Society Conference (FLAIRS)* (2005)
- Lang, T., Toussaint, M.: Approximate inference for planning in stochastic relational worlds. In: *Proceedings of the International Conference on Machine Learning (ICML)* (2009)
- Lang, T., Toussaint, M.: Probabilistic backward and forward reasoning in stochastic relational worlds. In: *Proceedings of the International Conference on Machine Learning (ICML)* (2010)
- Langley, P.: Cognitive architectures and general intelligent systems. *AI Magazine* 27, 33–44 (2006)

- Lanzi, P.L.: Learning classifier systems from a reinforcement learning perspective. *Soft Computing* 6, 162–170 (2002)
- Lecoeuche, R.: Learning optimal dialogue management rules by using reinforcement learning and inductive logic programming. In: Proceedings of the North American Chapter of the Association for Computational Linguistics, NAACL (2001)
- Letia, I., Precup, D.: Developing collaborative Golog agents by reinforcement learning. In: Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2001). IEEE Computer Society (2001)
- Levine, J., Humphreys, D.: Learning Action Strategies for Planning Domains Using Genetic Programming. In: Raidl, G.R., Cagnoni, S., Cardalda, J.J.R., Corne, D.W., Gottlieb, J., Guillot, A., Hart, E., Johnson, C.G., Marchiori, E., Meyer, J.-A., Middendorf, M. (eds.) EvoIASP 2003, EvoWorkshops 2003, EvoSTIM 2003, EvoROB/EvoRobot 2003, EvoCOP 2003, EvoBIO 2003, and EvoMUSART 2003. LNCS, vol. 2611, pp. 684–695. Springer, Heidelberg (2003)
- Lison, P.: Towards relational POMDPs for adaptive dialogue management. In: ACL 2010: Proceedings of the ACL 2010 Student Research Workshop, pp. 7–12. Association for Computational Linguistics, Morristown (2010)
- Littman, M.L., Sutton, R.S., Singh, S.: Predictive representations of state. In: Proceedings of the Neural Information Processing Conference (NIPS) (2001)
- Lloyd, J.W.: Logic for Learning: Learning Comprehensible Theories From Structured Data. Springer, Heidelberg (2003)
- Martin, M., Geffner, H.: Learning generalized policies in planning using concept languages. In: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR) (2000)
- Mausam, Weld, D.S.: Solving relational MDPs with first-order machine learning. In: Workshop on Planning under Uncertainty and Incomplete Information at ICAPS 2003 (2003)
- McCallum, R.A.: Instance-based utile distinctions for reinforcement learning with hidden state. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 387–395 (1995)
- Mellor, D.: A Learning Classifier System Approach to Relational Reinforcement Learning. In: Bacardit, J., Bernadó-Mansilla, E., Butz, M.V., Kovacs, T., Llorà, X., Takadama, K. (eds.) IWLCS 2006 and IWLCS 2007. LNCS (LNAI), vol. 4998, pp. 169–188. Springer, Heidelberg (2008)
- Minker, J.: Logic-Based Artificial Intelligence. Kluwer Academic Publishers Group, Dordrecht (2000)
- Minton, S., Carbonell, J., Knoblock, C.A., Kuokka, D.R., Etzioni, O., Gil, Y.: Explanation-based learning: A problem solving perspective. *Artificial Intelligence* 40(1-3), 63–118 (1989)
- Mooney, R.J., Califf, M.E.: Induction of first-order decision lists: Results on learning the past tense of english verbs. *Journal of Artificial Intelligence Research (JAIR)* 3, 1–24 (1995)
- Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13(1), 103–130 (1993)
- Morales, E.F.: Scaling up reinforcement learning with a relational representation. In: Proceedings of the Workshop on Adaptability in Multi-Agent Systems at AORC 2003, Sydney (2003)
- Morales, E.F.: Learning to fly by combining reinforcement learning with behavioral cloning. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 598–605 (2004)

- Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research (JAIR)* 11, 241–276 (1999)
- Mourão, K., Petrick, R.P.A., Steedman, M.: Using kernel perceptrons to learn action effects for planning. In: Proceedings of the International Conference on Cognitive Systems (CogSys), pp. 45–50 (2008)
- Muller, T.J., van Otterlo, M.: Evolutionary reinforcement learning in relational domains. In: Proceedings of the 7th European Workshop on Reinforcement Learning (2005)
- Nason, S., Laird, J.E.: Soar-RL: Integrating reinforcement learning with soar. In: Proceedings of the Workshop on Relational Reinforcement Learning at ICML 2004 (2004)
- Nath, A., Domingos, P.: A language for relational decision theory. In: International Workshop on Statistical Relational Learning, SRL (2009)
- Neruda, R., Slusny, S.: Performance comparison of two reinforcement learning algorithms for small mobile robots. *International Journal of Control and Automation* 2(1), 59–68 (2009)
- Oates, T., Cohen, P.R.: Learning planning operators with conditional and probabilistic effects. In: Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium, pp. 86–94 (1996)
- Pasula, H.M., Zettlemoyer, L.S., Kaelbling, L.P.: Learning probabilistic planning rules. In: Proceedings of the International Conference on Artificial Intelligence Planning Systems (ICAPS) (2004)
- Poole, D.: The independent choice logic for modeling multiple agents under uncertainty. *Artificial Intelligence* 94, 7–56 (1997)
- Ramon, J., Driessens, K., Croonenborghs, T.: Transfer Learning in Reinforcement Learning Problems Through Partial Policy Recycling. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenić, D., Skowron, A. (eds.) ECML 2007. LNCS (LNAI), vol. 4701, pp. 699–707. Springer, Heidelberg (2007)
- Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. The MIT Press, Cambridge (2001)
- Rodrigues, C., Gerard, P., Rouveiro, C.: On and off-policy relational reinforcement learning. In: Late-Breaking Papers of the International Conference on Inductive Logic Programming (2008)
- Rodrigues, C., Gérard, P., Rouveiro, C.: IncremEntal Learning of Relational Action Models in Noisy Environments. In: Frasconi, P., Lisi, F.A. (eds.) ILP 2010. LNCS, vol. 6489, pp. 206–213. Springer, Heidelberg (2011)
- Roncaglioni, S., Tadepalli, P.: Function approximation in hierarchical relational reinforcement learning. In: Proceedings of the Workshop on Relational Reinforcement Learning at ICML (2004)
- Russell, S.J., Norvig, P.: Artificial Intelligence: a Modern Approach, 2nd edn. Prentice Hall, New Jersey (2003)
- Ryan, M.R.K.: Using abstract models of behaviors to automatically generate reinforcement learning hierarchies. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 522–529 (2002)
- Saad, E.: A Logical Framework to Reinforcement Learning Using Hybrid Probabilistic Logic Programs. In: Greco, S., Lukasiewicz, T. (eds.) SUM 2008. LNCS (LNAI), vol. 5291, pp. 341–355. Springer, Heidelberg (2008)
- Safaei, J., Ghassem-Sani, G.: Incremental learning of planning operators in stochastic domains. In: Proceedings of the International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), pp. 644–655 (2007)

- Sanner, S.: Simultaneous learning of structure and value in relational reinforcement learning. In: Driessens, K., Fern, A., van Otterlo, M. (eds.) Proceedings of the ICML-2005 Workshop on Rich Representations for Reinforcement Learning (2005)
- Sanner, S.: Online feature discovery in relational reinforcement learning. In: Proceedings of the ICML-2006 Workshop on Open Problems in Statistical Relational Learning (2006)
- Sanner, S., Boutilier, C.: Approximate linear programming for first-order MDPs. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI) (2005)
- Sanner, S., Boutilier, C.: Practical linear value-approximation techniques for first-order MDPs. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI) (2006)
- Sanner, S., Boutilier, C.: Approximate solution techniques for factored first-order MDPs. In: Proceedings of the International Conference on Artificial Intelligence Planning Systems (ICAPS) (2007)
- Sanner, S., Kersting, K.: Symbolic dynamic programming for first-order pomdps. In: Proceedings of the National Conference on Artificial Intelligence (AAAI) (2010)
- Schmid, U.: Inductive synthesis of functional programs: Learning domain-specific control rules and abstraction schemes. In: Habilitationsschrift, Fakultät IV, Elektrotechnik und Informatik, Technische Universität Berlin, Germany (2001)
- Schuermans, D., Patrascu, R.: Direct value approximation for factored MDPs. In: Proceedings of the Neural Information Processing Conference (NIPS) (2001)
- Shapiro, D., Langley, P.: Separating skills from preference. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 570–577 (2002)
- Simpkins, C., Bhat, S., Isbell, C.L., Mateas, M.: Adaptive Programming: Integrating Reinforcement Learning into a Programming Language. In: Proceedings of the Twenty-Third ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA (2008)
- Slaney, J., Thiébaux, S.: Blocks world revisited. *Artificial Intelligence* 125, 119–153 (2001)
- Song, Z.W., Chen, X.P.: States evolution in $\Theta(\lambda)$ -learning based on logical mdps with negation. In: IEEE International Conference on Systems, Man and Cybernetics, pp. 1624–1629 (2007)
- Song, Z.W., Chen, X.P.: Agent learning in relational domains based on logical mdps with negation. *Journal of Computers* 3(9), 29–38 (2008)
- Stone, P.: Learning and multiagent reasoning for autonomous agents. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Computers and Thought Award Paper (2007)
- Stracuzzi, D.J., Asgharbeygi, N.: Transfer of knowledge structures with relational temporal difference learning. In: Proceedings of the ICML 2006 Workshop on Structural Knowledge Transfer for Machine Learning (2006)
- Sutton, R.S., Barto, A.G.: Reinforcement Learning: an Introduction. The MIT Press, Cambridge (1998)
- Sutton, R.S., McAllester, D.A., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Proceedings of the Neural Information Processing Conference (NIPS), pp. 1057–1063 (2000)
- Thielscher, M.: Introduction to the Fluent Calculus. *Electronic Transactions on Artificial Intelligence* 2(3-4), 179–192 (1998)
- Thon, I., Guttman, B., van Otterlo, M., Landwehr, N., De Raedt, L.: From non-deterministic to probabilistic planning with the help of statistical relational learning. In: Workshop on Planning and Learning at ICAPS (2009)

- Torrey, L.: Relational transfer in reinforcement learning. PhD thesis, University of Wisconsin-Madison, Computer Science Department (2009)
- Torrey, L., Shavlik, J., Walker, T., Maclin, R.: Relational macros for transfer in reinforcement learning. In: Proceedings of the International Conference on Inductive Logic Programming (ILP) (2007)
- Torrey, L., Shavlik, J., Natarajan, S., Kuppili, P., Walker, T.: Transfer in reinforcement learning via markov logic networks. In: Proceedings of the AAAI-2008 Workshop on Transfer Learning for Complex Tasks (2008)
- Toussaint, M.: Probabilistic inference as a model of planned behavior. *Künstliche Intelligenz (German Artificial Intelligence Journal)* 3 (2009)
- Toussaint, M., Plath, N., Lang, T., Jetchev, N.: Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference. In: IEEE International Conference on Robotics and Automation, ICRA (2010)
- Van den Broeck, G., Thon, I., van Otterlo, M., De Raedt, L.: DTProbLog: A decision-theoretic probabilistic prolog. In: Proceedings of the National Conference on Artificial Intelligence (AAAI) (2010)
- van Otterlo, M.: Efficient reinforcement learning using relational aggregation. In: Proceedings of the Sixth European Workshop on Reinforcement Learning, Nancy, France (EWRL-6) (2003)
- van Otterlo, M.: Reinforcement learning for relational MDPs. In: Nowé, A., Lenaerts, T., Steenhaut, K. (eds.) Machine Learning Conference of Belgium and the Netherlands (BeNeLearn 2004), pp. 138–145 (2004)
- van Otterlo, M.: Intensional dynamic programming: A rosetta stone for structured dynamic programming. *Journal of Algorithms* 64, 169–191 (2009a)
- van Otterlo, M.: The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for Adaptive Sequential Decision Making under Uncertainty in First-Order and Relational Domains. IOS Press, Amsterdam (2009b)
- van Otterlo, M., De Vuyst, T.: Evolving and transferring probabilistic policies for relational reinforcement learning. In: Proceedings of the Belgium-Netherlands Artificial Intelligence Conference (BNAIC), pp. 201–208 (2009)
- van Otterlo, M., Wiering, M.A., Dastani, M., Meyer, J.J.: A characterization of sapient agents. In: Mayorga, R.V., Perlovsky, L.I. (eds.) *Toward Computational Sapience: Principles and Systems*, ch. 9. Springer, Heidelberg (2007)
- Vargas, B., Morales, E.: Solving navigation tasks with learned teleo-reactive programs, pp. 4185–4185 (2008), doi:10.1109/IROS.2008.4651240
- Vargas-Govea, B., Morales, E.: Learning Relational Grammars from Sequences of Actions. In: Bayro-Corrochano, E., Eklundh, J.-O. (eds.) CIARP 2009. LNCS, vol. 5856, pp. 892–900. Springer, Heidelberg (2009)
- Vere, S.A.: Induction of relational productions in the presence of background information. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 349–355 (1977)
- Walker, T., Shavlik, J., Maclin, R.: Relational reinforcement learning via sampling the space of first-order conjunctive features. In: Proceedings of the Workshop on Relational Reinforcement Learning at ICML 2004 (2004)
- Walker, T., Torrey, L., Shavlik, J., Maclin, R.: Building relational world models for reinforcement learning. In: Proceedings of the International Conference on Inductive Logic Programming (ILP) (2007)
- Walsh, T.J.: Efficient learning of relational models for sequential decision making. PhD thesis, Rutgers University, Computer Science Department (2010)

- Walsh, T.J., Littman, M.L.: Efficient learning of action schemas and web-service descriptions. In: Proceedings of the National Conference on Artificial Intelligence (AAAI) (2008)
- Walsh, T.J., Li, L., Littman, M.L.: Transferring state abstractions between mdps. In: ICML-2006 Workshop on Structural Knowledge Transfer for Machine Learning (2006)
- Wang, C.: First-order markov decision processes. PhD thesis, Department of Computer Science, Tufts University, U.S.A (2007)
- Wang, C., Khordon, R.: Policy iteration for relational mdps. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI) (2007)
- Wang, C., Khordon, R.: Relational partially observable mdps. In: Proceedings of the National Conference on Artificial Intelligence (AAAI) (2010)
- Wang, C., Schmolze, J.: Planning with pomdps using a compact, logic-based representation. In: Proceedings of the IEEE International Conference on Tools with Artificial Intelligence, ICTAI (2005)
- Wang, C., Joshi, S., Khordon, R.: First order decision diagrams for relational MDPs. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (2007)
- Wang, C., Joshi, S., Khordon, R.: First order decision diagrams for relational MDPs. Journal of Artificial Intelligence Research (JAIR) 31, 431–472 (2008a)
- Wang, W., Gao, Y., Chen, X., Ge, S.: Reinforcement Learning with Markov Logic Networks. In: Gelbukh, A., Morales, E.F. (eds.) MICAI 2008. LNCS (LNAI), vol. 5317, pp. 230–242. Springer, Heidelberg (2008b)
- Wang, X.: Learning by observation and practice: An incremental approach for planning operator acquisition. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 549–557 (1995)
- Wingate, D., Soni, V., Wolfe, B., Singh, S.: Relational knowledge with predictive state representations. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (2007)
- Wooldridge, M.: An introduction to MultiAgent Systems. John Wiley & Sons Ltd., West Sussex (2002)
- Wu, J.H., Givan, R.: Discovering relational domain features for probabilistic planning. In: Proceedings of the International Conference on Artificial Intelligence Planning Systems (ICAPS) (2007)
- Wu, K., Yang, Q., Jiang, Y.: ARMS: Action-relation modelling system for learning action models. In: Proceedings of the National Conference on Artificial Intelligence (AAAI) (2005)
- Xu, J.Z., Laird, J.E.: Instance-based online learning of deterministic relational action models. In: Proceedings of the International Conference on Machine Learning (ICML) (2010)
- Yoon, S.W., Fern, A., Givan, R.: Inductive policy selection for first-order MDPs. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI) (2002)
- Zettlemoyer, L.S., Pasula, H.M., Kaelbling, L.P.: Learning planning rules in noisy stochastic worlds. In: Proceedings of the National Conference on Artificial Intelligence (AAAI) (2005)
- Zhao, H., Doshi, P.: Haley: A hierarchical framework for logical composition of web services. In: Proceedings of the International Conference on Web Services (ICWS), pp. 312–319 (2007)
- Zhuo, H., Li, L., Bian, R., Wan, H.: Requirement Specification Based on Action Model Learning. In: Huang, D.-S., Heutte, L., Loog, M. (eds.) ICIC 2007. LNCS, vol. 4681, pp. 565–574. Springer, Heidelberg (2007)

Chapter 9

Hierarchical Approaches

Bernhard Hengst

Abstract. Hierarchical decomposition tackles complex problems by reducing them to a smaller set of interrelated problems. The smaller problems are solved separately and the results re-combined to find a solution to the original problem. It is well known that the naïve application of reinforcement learning (RL) techniques fails to scale to more complex domains. This Chapter introduces hierarchical approaches to reinforcement learning that hold out the promise of reducing a reinforcement learning problems to a manageable size. Hierarchical Reinforcement Learning (HRL) rests on finding good re-usable temporally extended actions that may also provide opportunities for state abstraction. Methods for reinforcement learning can be extended to work with abstract states and actions over a hierarchy of subtasks that decompose the original problem, potentially reducing its computational complexity. We use a four-room task as a running example to illustrate the various concepts and approaches, including algorithms that can automatically learn the hierarchical structure from interactions with the domain.

9.1 Introduction

Artificial intelligence (AI) is about how to construct agents that act rationally. An agent acts rationally when it maximises a performance measure given a sequence of perceptions (Russell and Norvig, 1995). Planning and control theory can also be viewed from an agent perspective and included in this problem class. Reinforcement learning may at first seem like a seductive approach to solve the *artificial general intelligence (AGI)* problem (Hutter, 2007). While in principle this may be true, reinforcement learning is beleaguered by the “curse of dimensionality”. The curse of dimensionality is a term coined by Bellman (1961) to refer to the exponential

Bernhard Hengst
School of Computer Science and Engineering, University of New South Wales,
Sydney, Australia
e-mail: bernhardh@cse.unsw.edu.au

increase in the state-space with each additional variable or dimension that describes the problem. Bellman noted that sheer enumeration will not solve problems of any significance. It is unlikely that complex problems can be described by only a few variables, and so, it may seem we are at an impasse.

Fortunately the real world is highly structured with many constraints and with most parts independent of most other parts. Without structure it would be impossible to solve complex problems of any size (Russell and Norvig, 1995). Structure can significantly reduce the naïve state space generated by sheer enumeration. For example, if the transition and reward functions for two variables are independent, then a reinforcement learner would only need to explore a state-space the size of the addition of the state-space sizes of the variables instead of one the size of their product.

Reinforcement learning is concerned with problems represented by actions as well as states, generalising problem solving to systems that are dynamic in time. Hence we often refer to reinforcement learning problems as tasks, and sub-problems as sub-tasks.

This Chapter is concerned with leveraging hierarchical structure to try to reduce and solve more complex reinforcement learning problems that would otherwise be difficult if not impossible to solve.

Hierarchy

Empirically, a large proportion of complex systems seen in nature, exhibit hierarchical structure. By hierarchy we mean that the system is composed of interrelated sub-systems, that in turn have their own subsystems, and so on. Human societies have used hierarchical organisations to solve complex tasks dating back to at least Egyptian times. Hierarchical systems can be characterised as *nearly decomposable*, meaning that intra-component linkages are generally stronger than inter-component linkages (Simon, 1996). The property of near decomposability can help simplify their behaviour and description.

A heuristic from Polya (1945) for problem solving is *decomposing and recombining*, or *divide and conquer* in today's parlance. Many large problems have hierarchical structure that allows them to be broken down into sub-problems. The sub-problems, being smaller, are often solved more easily. The solutions to the sub-problems are recombined to provide the solution for the original larger problem. The decomposition can make finding the final solution significantly more efficient with improvements in the time and space complexity for both learning and execution.

There is another reason that hierarchy can simplify problem solving and make the task of learning more efficient. It is often the case that similar tasks need to be executed in different contexts. If the reinforcement learner is unaware of the underlying structure, it would relearn the same task in multiple different contexts, when it is clearly better to learn the task once and reuse it. Programmers use function calls and subroutines for just this purpose – to avoid repeating similar code

fragments. In many ways *hierarchical reinforcement learning (HRL)* is about structuring reinforcement learning problems much like a computer program where subroutines are akin to subtasks of a higher-level reinforcement learning problem. Just as the main program calls subroutines, a higher level reinforcement problem invokes subtasks.

Four-Room Task

Throughout this Chapter we will use a simple four-room task as a running example to help illustrate concepts. Figure 9.1 (left) shows the agent view of reinforcement learning. The agent interacts with the environment in a sense-act loop, receiving a reward signal at each time-step as a part of the input.

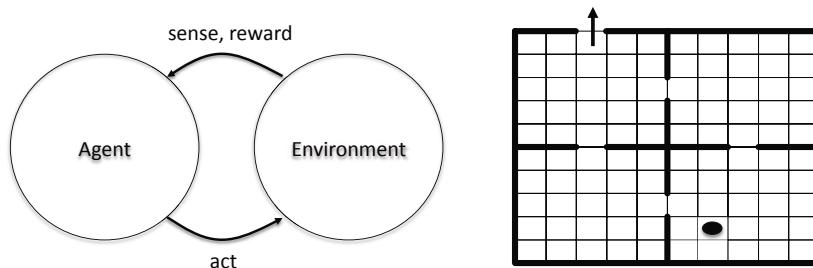


Fig. 9.1 Left: The agent view of Reinforcement Learning. Right: A four-room task with the agent in one of the rooms shown as a solid black oval.

In the four-room example, the agent is represented as a black oval at a grid location in the South-East room of a four-room house (Figure 9.1, right). The rooms are connected by open doorways. The North-West room has a doorway leading out of the house. At each time-step the agent takes an action and receives a sensor observation and a reward from the environment.

Each cell represents a possible agent position. The position is uniquely described by each room and the position in each room. In this example the rooms have similar dimensions and similar positions in each room are assumed to be described by the same identifier. The environment is *fully observable* by the agent which is able to sense both the room it occupies and its position in the room. The agent can move one cell-step in any of the four compass directions at each time-step. It also receives a reward of -1 at each time-step. The objective is to leave the house via the least-cost route. We assume that the actions are stochastic. When an action is taken there is an 80% chance that the agent will move in the intended direction and a 20% chance that it will stay in place. If the agent moves into a wall it will remain where it is.

Hierarchical Reinforcement Learning (HRL)

Our approach to HRL starts with a well specified reinforcement learning problem modelled as an Markov Decision Process (MDP) as described in Chapter 1. The reader can easily verify that the four-room task is such a reinforcement learning problem. We provide here an initial intuitive description of how HRL methods might be applied to the four-room task.

If we can find a policy to leave a room, say by the North doorway, then we could reuse this policy in any of the rooms because they are identical. The problem to leave a room by the North doorway is just another, albeit smaller, reinforcement learning problem that has inherited the position-in-room states, actions, transition and reward function from the original problem. We proceed to solve two smaller reinforcement learning problems, one to find a room-leaving policy to the North and another to leave a room through the West doorway.

We also formulate and solve a higher-level reinforcement learning problem that uses only the four room-states. In each room-state we allow a choice of executing one of the previously learnt room-leaving policies. For the higher-level problem these policies are viewed as *temporally extended* actions because once they are invoked they will usually persist for multiple time-steps until the agent exits a room. At this stage we simply specify a reward of -1 per room-leaving action. As we shall see later, reinforcement learning can be generalised to work with temporally extended actions using the formalism of semi-Markov Decision Processes (SMDP).

Once learnt, the execution of the higher-level house-leaving policy will determine the room-leaving action to invoke given the current room. Control is passed to the room-leaving sub-task that leads the agent out of the room through the chosen doorway. Upon leaving the room, the sub-task is terminated and control is passed back to the higher level that chooses the next room-leaving action until the agent finally leaves the house.

The above example hides many issues that HRL needs to address, including: safe state abstraction; appropriately accounting for accumulated sub-task reward; optimality of the solution; and specifying or even learning of the hierarchical structure itself. In the next sections we will discuss these issues and review several approaches to HRL.

9.2 Background

This section will introduce several concepts that are important to understanding hierarchical reinforcement learning (HRL). There is general agreement that temporally extended actions and the related semi-Markov Decision Process formalism are key ingredients. We will also discuss issues related to problem size reduction and solution optimality.

9.2.1 Abstract Actions

Approaches to HRL employ actions that persist for multiple time-steps. These temporally extended or *abstract actions* hide the multi-step state-transition and reward details from the time they are invoked until termination. The room-leaving actions discussed above for the four-room task are examples. Leaving a room may involve taking multiple single-step actions to first navigate to a doorway and then stepping through it.

Abstract actions are employed in many fields including AI, robotics and control engineering. They are similar to *macros* in computer science that make available a sequence of instructions as a single program statement. In planning, macros help decompose and solve problems (see for example solutions to the Fifteen Puzzle and Rubik's Cube (Korf, 1985)).

Abstract actions in an MDP setting extend macros in that the more primitive steps that comprise an abstract action may be modelled with stochastic transition functions and use stochastic policies (see Chapter 1). Abstract actions may execute a policy for a smaller Markov Decision Problem. Stochasticity can manifest itself in several ways. When executing an abstract action a stochastic transition function will make the sequence of states visited non-deterministic. The sequence of rewards may also vary, depending on the sequence of states visited, even if the reward function itself is deterministic. Finally, the time taken to complete an abstract action may vary. Abstract actions with deterministic effects are just classical macro operators.

Properties of abstract actions can be seen in the four-room task. In this problem the 20% chance of staying in situ, does not make it possible to determine beforehand how many time-steps it will take to leave a room when a room-leaving abstract action is invoked.

Abstract actions can be generalised for continuous-time (Puterman, 1994), however, this Chapter will focus on discrete-time problems. Special case abstract actions that terminate in one time-step are just ordinary actions and we refer to them as *primitive* actions.

9.2.2 Semi-Markov Decision Problems

We will now extend MDPs from Chapter 1 to MDPs that include abstract actions. MDPs that include abstract actions are called *semi Markov Decision Problems*, or SMDPs (Puterman, 1994). As abstract actions can take a random number of time-steps to complete, we need to introduce another variable to account for the time to termination of the abstract action.

We denote the random variable $N \geq 1$ to be the number of time steps that an abstract action a takes to complete, starting in state s and terminating in state s' .

The model of the SMDP, defined by the state transition probability function and the expected reward function now includes the random variable N .¹

The transition function $T : S \times A \times S \times N \rightarrow [0,1]$ gives the probability of the abstract action a terminating in state s' after N steps, having been initiated in state s .

$$T(s,a,s',N) = \Pr\{s_{t+N} = s' | s_t = s, a_t = a\} \quad (9.1)$$

The reward function for an abstract action accumulates single step rewards as it executes. The manner in which rewards are summed depends on the performance measure included with the SMDP. A common performance measure is to maximise the sum of future discounted rewards using a constant discount factor γ . The reward function $R : S \times A \times S \times N \rightarrow \mathbb{R}$ that gives the expected discounted sum of rewards when an abstract action a is started in state s and terminates in state s' after N steps is

$$R(s,a,s',N) = E \left\{ \sum_{n=0}^{N-1} \gamma^n r_{t+n} | s_t = s, a_t = a, s_{t+N} = s' \right\} \quad (9.2)$$

The value functions and Bellman “backup” equations from Chapter 1 for MDPs can also be generalised for SMDPs. The value of state s for policy π , denoted $V^\pi(s)$ is the expected return starting in state s at time t , and taking abstract actions according to π .²

$$V^\pi(s) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \right\}$$

If the abstract action executed in state s is $\pi(s)$, persists for N steps and terminates we can write the value function as two series – the sum of the rewards accumulated for the first N steps and the remainder of the series of rewards.

$$V^\pi(s) = E_\pi \left\{ (r_t + \gamma r_{t+1} + \dots + \gamma^{N-1} r_{t+N-1}) + (\gamma^N r_{t+N} + \dots) | s_t = s \right\}$$

Taking the expectation with respect to both s' and N with probabilities given by Equation 9.1, substituting the N -step reward for abstract action $\pi(s)$ from Equation 9.2, and recognising that the second series is just the value function starting in s' discounted by N steps, we can write

$$V^\pi(s) = \sum_{s',N} T(s,\pi(s),s',N) [R(s,\pi(s),s',N) + \gamma^N V^\pi(s')]$$

¹ This formulation of an SMDP is based on Sutton et al (1999) and Dietterich (2000), but has been changed to be consistent with notation in the Chapter 1.

² Note that we are overloading function π . $\pi(s,a)$ is the probability of choosing abstract action a in state s . $\pi(s)$ is the abstract action that is chosen in state s under a deterministic policy π .

The optimum value function for an SMDP (denoted by $*$) is also similar to that for MDPs with the sum taken with respect to s' and N .

$$V^*(s) = \max_a \sum_{s',N} T(s,a,s',N)[R(s,a,s',N) + \gamma^N V^*(s')]$$

Similarly, Q abstract action value functions can be written for SMDP policies. The definition of $Q^\pi(s,a)$ is the value of taking abstract action a in state s and then following the SMDP policy π thereafter. Hence

$$Q^\pi(s,a) = \sum_{s',N} T(s,a,s',N)[R(s,a,s',N) + \gamma^N Q^\pi(s',\pi(s'))] \quad (9.3)$$

The optimum SMDP value function is

$$Q^*(s,a) = \sum_{s',N} T(s,a,s',N)[R(s,a,s',N) + \gamma^N V^*(s')] \\ \text{where } V^*(s') = \max_{a'} Q^*(s',a')$$

For problems that are guaranteed to terminate, the discount factor γ can be set to 1. In this case the number of steps N can be marginalised out in the above equations and the sum taken with respect to s alone. The equations are then similar to the ones for MDPs with the expected primitive reward replaced with the expected sum of rewards to termination of the abstract action.

All the methods developed for solving Markov decision processes in the Chapter 1 for reinforcement learning using primitive actions work equally well for problems using abstract actions. As primitive actions are just a special case of abstract actions we include them in the set of abstract actions.

The reader may well wonder whether the introduction of abstract actions buys us anything. After all we have just added extra actions to the problem and increased the complexity. The rest of this Chapter will show how abstract actions allow us to leverage structure in problems to reduce storage requirements and increase the speed of learning.

In a similar four-room example to that of Figure 9.1, Sutton et al (1999) show how the presence of abstract actions allows the agent to learn significantly faster proceeding on a room by room basis, rather than position by position³. When the goal is not in a convenient location, able to be reached by the given abstract actions, it is possible to include primitive actions as special-case abstract actions and still accelerate learning for some problems. For example, with room-leaving abstract actions alone, it may not be possible to reach a goal in the middle of a room.

Unless we introduce other abstract actions, primitive actions are still required when the room containing the goal state is entered. Although the inclusion of primitive actions guarantees convergence to the globally optimal policy, this may create extra work for the learner. Reinforcement learning may be accelerated because the

³ In this example, abstract actions take the form of options and will be defined in Subsection 9.3.1

value function can be backed-up over greater distances in the state-space and the inclusion of primitive actions guarantees convergence to the globally optimal policy, but the introduction of additional actions increased the storage and exploration necessary.

9.2.3 Structure

Abstract actions and SMDPs naturally lead to hierarchical structure. With appropriate abstract actions alone we may be able to learn a policy with less effort than it would take to solve the problem using primitive actions. This is because abstract actions can skip over large parts of the state-space terminating in a small subset of states. We saw in the four-room task how room-leaving abstract actions are able to reduce the problem state-space to room states alone.

Abstract actions themselves may be policies from smaller SMDPs (or MDPs). This establishes a hierarchy where a higher-level *parent* task employs *child* subtasks as its abstract actions.

Task Hierarchies

The parent-child relationship between SMDPs leads to *task-hierarchies* (Dietterich, 2000). A task-hierarchy is a directed acyclic graph of sub-tasks. The root-node in the task-hierarchy is a top-level SMDP that can invoke its child-node SMDP policies as abstract actions. Child-node policies can recursively invoke other child subtasks, right down to sub-tasks that only invoke primitive actions. The designer judiciously specifies abstract actions that are available in states of each parent task, and specifies the active states and terminal states for each sub-task.

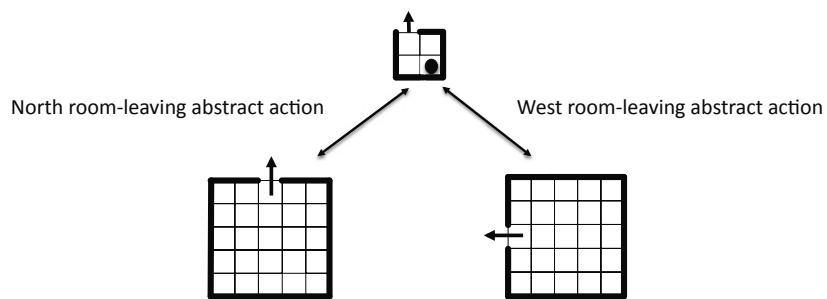


Fig. 9.2 A task-hierarchy decomposing the four-room task in Figure 9.1. The two lower-level sub-tasks are generic room-leaving abstract actions, one each for leaving a room to the North and West.

Figure 9.2 shows a task-hierarchy for the four-room task discussed in Section 9.1. The two lower-level sub-tasks are MDPs for a generic room, where separate policies are learnt to exit a room to the North and West. The arrows indicate transitions to terminal states. States, actions, transitions and rewards are inherited from the original MDP. The higher level problem (SMDP) consists of just four states representing the rooms. Any of the sub-tasks (room-leaving actions) can be invoked in any of the rooms.

Partial Programs

Manually specifying a task-hierarchy is a form of programming allowing the designer to include background knowledge to exploit the hierarchical structure of the problem. Not all sub-task policies need to be learnt. If a programmer already has knowledge of good sub-task polices these can be hand-coded and supplied in the form of stochastic finite state machines.

As computers are finite state machines and can approximate stochastic finite state machines, task-hierarchies may be able to be efficiently represented as partial programs using the full programming structure available. We will elaborate on this approach to HRL in Section 9.3.2.

9.2.4 State Abstraction

The benefit of decomposing a large MDP is that it will hopefully lead to *state abstraction* opportunities to help reduce the complexity of the problem. An abstracted state space is smaller than the state space of an original MDP. There are broadly two kinds of conditions under which state-abstractions can be introduced (Dietterich, 2000). They are situations in which:

- we can eliminate irrelevant variables, and
- where abstract actions “funnel” the agent to a small subset of states.

Eliminating Irrelevant Variables

When reinforcement learning algorithms are given redundant information they will learn the same value-function or policy for all the redundant states. For example, navigating through a red coloured room may be the same as for a blue coloured room, but a value function and policy treats each (*position-in-room, colour*) as a different state. If colour has no effect on navigation it would simplify the problem by eliminating the colour variable from consideration.

More generally, if we can find a partition of the state-space of a subtask m such that all the transitions from states in one block of the partition have the same probability and expected reward to transitioning to each of the other blocks, we can reduce

the subtask to one where the states become the blocks. The solution of this reduced subtask is the solution of the original subtask. This is the notion of *stochastic bisimulation homogeneity* for MDP model minimisation (Dean and Givan, 1997). The computational complexity of solving an MDP, being polynomial in $|S|$, will be reduced depending on the coarseness of the partition. The state abstraction can be substantial. Consider the subtask that involves walking. This skill is mostly independent of geography, dress, and objects in the world. If we could not abstract the walking subtask, we would be condemned to re-learn to walk every-time any one of these variables changed value.

Formally, if $P = \{B_1, \dots, B_n\}$ is a partition of the state-space of an SMDP and for each: $B_i, B_j \in P, a \in A, p, q \in B_i$, number of time-steps to termination is N , then if and only if

$$\begin{aligned}\sum_{r \in B_j} T(p, a, r, N) &= \sum_{r \in B_j} T(q, a, r, N) \\ \sum_{r \in B_j} R(p, a, r, N) &= \sum_{r \in B_j} R(q, a, r, N)\end{aligned}$$

the subtask can be minimised to one in which S is replaced by P ,

$$\begin{aligned}T(B_i, a, B_j, N) &= \sum_{r \in B_j} T(p, a, r, N) \\ R(B_i, a, B_j, N) &= \sum_{r \in B_j} R(p, a, r, N)\end{aligned}$$

Model minimisation is exemplified by the room-leaving abstract actions in the four-room task. For these subtasks the particular room is irrelevant and the room variable can be eliminated. In this case, the blocks of the partition of the total state-space comprise states with the same position-in-room value.

This type of state abstraction is used by Boutilier et al (1995) who exploit independence between variables in dynamic Bayes nets. It is introduced by Dietterich (2000) for Max Node and Leaf Irrelevance in the MAXQ graph (Section 9.3.3). Ravindran and Barto (2003) extend this form of model minimisation to state-action symmetry couched in the algebraic formalism of morphisms. They state conditions under which the minimised model is a homomorphic image and hence transitions and rewards commute.

Some states may not be reachable for some sub-tasks in the task-hierarchy. This Shielding condition (Dietterich, 2000) is another form of elimination of irrelevant variables, in this case resulting from the structure of the task-graph.

Funnelling

Funnelling (Dietterich, 2000) is a type of state abstraction where abstract actions move the environment from a large number of initial states to a small number of

resulting states. The effect is exploited for example by Forestier and Varaiya (1978) in plant control and by Dean and Lin (1995) to reduce the size of the MDP.

Funneling can be observed in the four-room task. Room-leaving abstract actions move the agent from any position in a room to the state outside the respective doorway. Funneling allows the four-room task to be state-abstraction at the root node to just 4 states because, irrespective of the starting position in each room, the abstract actions have the property of moving the agent to another room state.

9.2.5 Value-Function Decomposition

The task-hierarchy for the four-room task in Figure 9.2 has two successful higher-level policies that will find a path out of the house from the starting position in the South-East room. They are to leave rooms successively either North-West-North or West-North-North. The latter is the shorter path, but the simple hierarchical reinforcement learner in Section 9.1 cannot make this distinction.

What is needed is a way to decompose the value function for the whole problem over the task-hierarchy. Given this decomposition we can take into account the rewards within a subtask when making decision at higher levels.

To see this, consider the first decision that the agent in the four-room task (Figure 9.1) has to make. Deciding whether it is better to invoke a North or West room-leaving abstract action does not just depend on the agent's room state – the South-West room in this case. It also depends on the current position in the room. In this case we should aim for the doorway that is closer, as once we have exited the room, the distance out of the house is the same for either doorway. We need both the room-state and the position-in-room state to decide on the best action.

The question now arises as to how to decompose the original value function given the task-hierarchy so that the optimal action can be determined in each state. This will be addressed by the MAXQ approach (Dietterich, 2000) in Section 9.3.3 with a two part decomposition of the value function for each subtask in the task-hierarchy. The two parts are value to termination of the abstract action and the value to termination of the subtask.

Andre and Russell (2002) have introduced a three part decomposition of the value function by including the component of the value to complete the problem after subtask terminates. The advantage of this approach is that context is taking into account, making solutions hierarchically optimal (see Section 9.2.6), but usually at the expense of less state abstraction. The great benefit of decomposition is state-abstraction opportunities.

9.2.6 Optimality

We are familiar with the notion of an optimal policy for an MDP from Chapter 1. Unfortunately, HRL cannot guarantee in general that a decomposed problem will necessarily yield the optimal solution. It depends on the problem and the quality of

the decomposition in terms of the abstract actions available and the structure of the task hierarchy or the partial program.

Hierarchically Optimal

Policies that are *hierarchically optimal* are ones that maximise the overall value function consistent with the constraints imposed by the task-hierarchy. To illustrate this concept assume that the agent in the four-room task moves with a 70% probability in the intended direction, but slips with a 10% probability each of the other three directions. Executing the hierarchical optimal policy for the task-hierarchy shown in Figure 9.2 may not be optimal. The top level policy will choose the West room-leaving action to leave the room by the nearest doorway. If the agent should find itself near the North doorway due to stochastic drift, it will nevertheless stubbornly persist to leave by the West doorway as dictated by the policy of the West room-leaving abstract action. This is suboptimal. A “flat” reinforcement learner executing the optimal policy would change its mind and attempt to leave the South-West room by the closest doorway.

The task-hierarchy could once again be made to yield the optimal solution if we included an abstract action that was tasked to leave the room by either the West or North doorway.

Recursively Optimal

Dietterich (2000) introduced another form of optimality - *recursive optimality*. In this type of optimality sub-task policies to reach goal terminal states are *context free* ignoring the needs of their parent tasks. This formulation has the advantage that sub-tasks can be re-used in various contexts, but they may not therefore be optimal in each situation.

A recursively optimal solution cannot be better than a hierarchical optimal solution. A hierarchical optimality can be arbitrarily worse than the globally optimal solution and a recursive optimal solution can be arbitrarily worse than a hierarchical optimal solution. It depends on the design of the task hierarchy by the designer.

Hierarchical Greedy Optimality

As we have seen above the stochastic nature of MDPs means that the condition under which an abstract action is appropriate may have changed after the action’s invocation and that another action may become a better choice because of the inherent stochasticity of transitions (Hauskrecht et al, 1998). A subtask policy proceeding to termination may be sub-optimal. By constantly interrupting the sub-task a better sub-task may be chosen. Dietterich calls this “polling” procedure *hierarchical greedy execution*. While this is guaranteed to be no worse than a hierarchical optimal solution or a recursively optimal solution and may be considerably better, it still does not provide any global optimality guarantees.

Hauskrecht et al (1998) discuss decomposition and solution techniques that make optimality guarantees, but unfortunately, unless the MDP can be decomposed into weakly coupled smaller MDPs the computational complexity is not necessarily reduced.

9.3 Approaches to Hierarchical Reinforcement Learning (HRL)

HRL continues to be an active research area. We will now briefly survey some of the work in HRL. Please also see Barto and Mahadevan (2003) for a survey of advances in HRL, Si et al (2004) for hierarchical decision making and approaches to concurrency, multi-agency and partial observability, and Ryan (2004) for an alternative treatment and motivation underlying the movement towards HRL.

Historical Perspective

Ashby (1956) talks about amplifying the regulation of large systems in a series of stages, describing these hierarchical control systems “ultra-stable”. HRL can be viewed as a gating mechanisms that, at a higher levels, learn to switch in appropriate and more reactive behaviours at lower levels. Ashby (1952) proposed such a gating mechanism for an agent to handle recurrent situations.

The subsumption architecture (Brooks, 1990) works along similar lines. It decomposes complicated behaviour into many simple tasks which are organised into layers, with higher level layers becoming increasingly abstract. Each layer’s goal subsumes that of its child layers. For example, obstacle avoidance is subsumed by a foraging for food parent task and switched in when an obstacle is sensed in the robot’s path.

Watkins (1989) discusses the possibility of hierarchical control consisting of coupled Markov decision problems at each level. In his example, the top level, like the navigator of an 18th century ship, provides a kind of gating mechanism, instructing the helmsman on which direction to sail. Singh (1992) developed a gating mechanism called Hierarchical-DYNA (H-DYNA). DYNA is a reinforcement learner that uses both real and simulated experience after building a model of the reward and state transition function. H-DYNA first learns elementary tasks such as to navigate to specific goal locations. Each task is treated as an abstract action at a higher level of control.

The “feudal” reinforcement learning algorithm (Dayan and Hinton, 1992) emphasises another desirable property of HRL - state abstraction. The authors call it “information hiding”. It is the idea that decision models should be constructed at coarser granularities further up the control hierarchy.

In the hierarchical distance to goal (HDG) algorithm, Kaelbling (1993) introduces the important idea of composing the value function from distance components along the path to a goal. HDG is modelled on navigation by landmarks. The idea is

to learn and store local distances to neighbouring landmarks and distances between any two locations within each landmark region. Another function is used to store shortest-distance information between landmarks as it becomes available from local distance functions. The HDG algorithm aims for the next nearest landmark on the way to the goal and uses the local distance function to guide its primitive actions. A higher level controller switches lower level policies to target the next neighbouring landmark whenever the agent enters the last targeted landmark region. The agent therefore rarely travels through landmarks but uses them as points to aim for on its way to the goal. This algorithm provided the inspiration for both the MAXQ value function decomposition (to be discussed in more detail in Section 9.3.3) and value function improvement with hierarchical greedy execution (Paragraph 9.2.6).

Moore et al (1999) have extended the HDG approach with the “airport-hierarchy” algorithm. The aim is to find an optimal policy to move from a start state to a goal state where both states can be selected from the set of all states. This *multi-goal* problem requires an MDP with $|S|^2$ states. By learning a set of goal-state reaching abstract actions with progressively smaller “catchment” areas, it is possible to approximate an optimal policy to any goal-state using, in the best case, only order $N \log N$ states.

Abstract actions may be included in hybrid hierarchies with other learning or planning methods mixed at different levels. For example, Ryan and Reid (2000) use a hybrid approach (RL-TOP) to constructing task-hierarchies. In RL-TOP, planning is used at the abstract level to invoke reactive planning operators, extended in time, based on teleo-reactive programs (Nilsson, 1994). These operators use reinforcement learning to achieve their post-conditions as sub-goals.

Three paradigms predominate HRL. They are: *Options*, a formalisation of abstract actions; HAMQ, a partial program approach, and MAXQ value-function decomposition including state abstraction.

9.3.1 Options

One formalisation of an abstract action is the idea of an *option* (Sutton et al, 1999)⁴.

Definition 9.3.1. An *option* (in relation to an MDP $\langle S, A, T, R \rangle$) is a triple $\langle I, \pi, \beta \rangle$ in which $I \subseteq S$ is an initiation set, $\pi : S \times A \rightarrow [0,1]$ is a policy, and $\beta : S^+ \rightarrow [0,1]$ is a termination condition.

An option can be taken in state $s \in S$ of the MDP if $s \in I$. This allows option initiation to be restricted to a subset of S . Once invoked the option takes actions as determined by the stochastic policy π . Options terminate stochastically according to function β that specifies the probability of termination in each state. Many tasks are *episodic*, meaning that they will eventually terminate. When we include a single

⁴ Abstract actions can be formalised in slightly different ways. One such alternative is used with the HEXQ algorithm to be described later.

abstract terminal state in the definition of the state space of an MDP we write S^+ to denote the total state space including the terminal abstract state. If the option is in state s it will terminate with probability $\beta(s)$, otherwise it will continue execution by taking the next action a with probability $\pi(s,a)$.

When option policies and termination depend on only the current state s , options are called *Markov options*. We may wish to base a policy on information other than the current state s . For example, if we want the option to time-out after a certain number of time-steps we could add a counter to the state space. In general the option policy and termination can depend on the entire history sequence of states, actions and rewards since the option was initiated. Options of this sort are called *semi-Markov* options. We will later discuss examples of options that are formulated by augmenting an MDP by a stochastic finite state machine, such as a program.

It is easy to see that a primitive action is an option. A primitive action a is equivalent to the option $\langle I = s, \pi(s,a) = 1.0, \beta(s) = 1 \rangle$ for all $s \in S$. It is possible to unify the set of options and primitive actions and take set A to be their union.

Just as with policies over primitive actions, we can define policies over options. The option policy $\pi : S \times A \rightarrow [0,1]$ is a function that selects an option with probability $\pi(s,a)$, $s \in S, a \in A$. Since options select actions, and actions are just special kinds of options, it is possible for options to select other options. In this way we can form hierarchical structures to an arbitrary depth.

9.3.2 HAMQ-Learning

If a small hand-coded set of abstract actions are known to help solve an MDP we may be able to learn a policy with much less effort than it would take to solve the problem using primitive actions. This is because abstract actions can skip over large parts of the state space terminating in a small subset of states. The original MDP may be made smaller because now we only need to learn a policy with the set of abstract actions over a reduced state space.

In the *hierarchy of abstract machines* (HAM) approach to HRL the designer specifies abstract actions by providing stochastic finite state automata called *abstract machines* that work jointly with the MDP (Parr and Russell, 1997). This approach explicitly specifies abstract actions allowing users to provide background knowledge, more generally in the form of partial programs, with various levels of expressivity.

An abstract machine is a triple $\langle \mu, I, \delta \rangle$, where μ is a finite set of machine states, I is a stochastic function from states of the MDP to machine states that determines the initial machine state, and δ is a stochastic next-state function, mapping machine states and MDP states to next machine states. The machine states are of different types. Action-states specify the action to be taken given the state of the MDP to be solved. Call-states execute another machine as a subroutine. Choice-states non-deterministically select the next machine state. Halt-states halt the machine.

The parallel action of the abstract machine and the MDP yields a discrete-time higher-level SMDP. Choice-states are states in which abstract actions can be initiated. The abstract machine's action-states and choice states generate a sequence of actions that amount to an abstract action policy. If another choice-state is reached before the current executing machine has terminated, this is equivalent to an abstract action selecting another abstract action, thereby creating another level in a hierarchy. The abstract action is terminated by halt-states. A judicious specification of a HAM may reduce the set of states of the original MDP associated with choice-points.

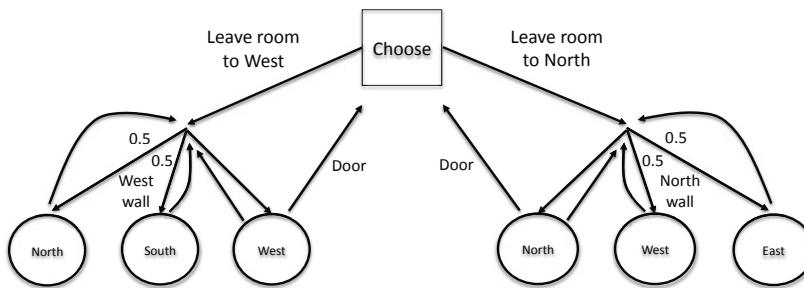


Fig. 9.3 An abstract machine (HAM) that provides routines for leaving rooms to the West and North of the house in Figure 9.1. The choice is between two abstract actions. One to leave the house through a West doorway, the other through a North doorway. If the West doorway is chosen, the abstract action keeps taking a West primitive action until it has either moved though the door and terminates or reaches a West wall. If at a West wall it takes only North and South primitive actions at random until the West wall is no longer observable, i.e. it has reached a doorway, whereupon it moves West through the doorway and terminates.

We can illustrate the operation of a HAM using the four-room example. The abstract machine in Figure 9.3 provides choices for leaving a room to the West or the North. In each room it will take actions that move the agent to a wall, and perform a random walk along the wall until it finds the doorway. Primitive rewards are summed between choice states. In this example we assume the agent's initial position is as shown in Figure 9.1. Only five states of the original MDP are states of the SMDP. These states are the initial state of the agent and the states on the other side of doorways where the abstract machine enters choice states. Reinforcement learning methods update the value function for these five states in the usual way with rewards accumulated since the last choice state.

Solving the SMDP yields an optimal policy for the agent to leave the house subject to the program constraints of the abstract machine. The best policy consists of the three abstract actions – sequentially leaving a room to the West, North and North again. In this case it is not a globally optimal policy because a random walk

along walls to find a doorway is inefficient. The HAM approach is predicated on engineers and control theorists being able to design good controllers that will realise specific behaviours. HAMs are a way to partially specify procedural knowledge to transform an MDP to a reduced SMDP.

Programmable HRL

In the most general case a HAM can be a program that executes any computable mapping of the agent’s complete sensory-action history (Parr, 1998).

Andre and Russell extended the HAM approach by introducing more expressive agent design languages for HRL – Programmable HAM (PHAM) (Andre and Russell, 2000) and ALisp, a Lisp-based high-level partial programming language (Andre and Russell, 2002).

Golog, is a logic programming language for agents that allow agents to reason about actions, goals, perceptions, other agents, etc., using situation calculus (Levesque et al, 1997). It has been extended as a partial program by embedding MDPs. Examples include *decision theoretic* Golog (DTGolog) (Boutilier et al, 2000) and Readylog (Ferrein and Lakemeyer, 2008) using the options framework.

In each case the partial program allows users to provide background knowledge about the problem structure using special choice-point routines that implement non-deterministic actions for the agent to learn the best action to take from experience. Programs of this kind leverage the expressiveness of the programming language to succinctly specify (and solve) an SMDP.

9.3.3 MAXQ

MAXQ is an approach to HRL where the value function is decomposed over the task hierarchy (Dietterich, 2000). It can lead to a compact representation of the value function and makes sub-tasks *context-free* or portable.

MAXQ abstract actions are crafted by classifying subtask terminal states as either goal states or non-goal states. Using disincentives for non-goal states, policies are learnt for each subtask to encourage termination in goal states. This termination predicate method may introduce an additional source of sub-optimality in the MDP as “pseudo” rewards can distort the subtask policy.

A key feature of MAXQ is that it represents the value of a state as a decomposed sum of sub-task completion values plus the expected reward for the immediate primitive action. A completion value is the expected (discounted) cumulative reward to complete the sub-task after taking the next abstract action.

We will derive the hierarchical decomposition following Dietterich (2000) and extend the above SMDP notation by including explicit reference to a particular subtask m . Equation 9.3 for subtask m becomes:

$$\begin{aligned} Q^\pi(m,s,a) = & \sum_{s',N} T(m,s,a,s',N)[R(m,s,a,s',N) \\ & + \gamma^N Q^\pi(m,s',\pi(s'))] \end{aligned} \quad (9.4)$$

Abstract action a for subtask m invokes a child subtask m_a . The expected value of completing subtask m_a is expressed as $V^\pi(m_a,s)$. The *hierarchical policy*, π , is a set of policies, one for each subtask.

The *completion function*, $C^\pi(m,s,a)$, is the expected discounted cumulative reward after completing abstract action a , in state s in subtask m , discounted back to the point where a begins execution.

$$C^\pi(m,s,a) = \sum_{s',N} T(m,s,a,s',N)\gamma^N Q^\pi(m,s',\pi(s'))$$

The Q function for subtask m (Equation 9.4) can be expressed recursively as the value for completing the subtask that a invokes, m_a , plus the completion value to the end of subtask m .

$$Q^\pi(m,s,a) = V^\pi(m_a,s) + C^\pi(m,s,a) \quad (9.5)$$

The value of completing subtask m_a depends on whether a is primitive or not. In the event that action a is primitive, $V^\pi(m_a,s)$ is the expected reward after taking action a in state s .

$$V^\pi(m_a,s) = \begin{cases} Q^\pi(m_a,s,\pi(s)) & \text{if } a \text{ is abstract} \\ \sum_{s'} T(s,a,s')R(s,a,s') & \text{if } a \text{ is primitive} \end{cases} \quad (9.6)$$

If the path of activated subtasks from root subtask m_0 to primitive action m_k is m_0, m_1, \dots, m_k , and the hierarchical policy specifies that in subtask m_i , $\pi(s) = a_i$, then recursive Equations 9.5 and 9.6 decompose the value function $Q^\pi(m_0,s,\pi(s))$ as

$$\begin{aligned} Q^\pi(m_0,s,\pi(s)) = & V^\pi(m_k,s) + C^\pi(m_{k-1},s,a_{k-1}) + \dots \\ & + C^\pi(m_1,s,a_1) + C^\pi(m_0,s,a_0) \end{aligned}$$

To follow an optimal greedy policy given the hierarchy, the decomposition Equation 9.6 for the subtask implementing abstract action a is modified to choose the best action a , i.e. $V^*(m_a,s) = \max_{a'} Q^*(m_a,s,a')$. The introduction of the *max* operator means that we have to perform a complete search through all the paths in the task-hierarchy to determine the best action. Algorithm 18 performs such a depth-first search and returns both the value and best action for subtask m in state s .

As the depth of the task-hierarchy increases, this exhaustive search can become prohibitive. Limiting the depth of the search is one way to control its complexity (Hengst, 2004). To plan an international trip, for example, the flight and airport-transfer methods need to be considered, but optimising which side of the bed to get out of on the way to the bathroom on the day of departure can effectively be ignored for higher level planning.

```

Require:  $V(m,s)$  for primitive actions  $m$ , abstract actions  $A_m$ ,  $C(m,s,j) \forall j \in A_m$ 
1: if  $m$  is a primitive action then
2:   return  $\langle V(m,s), m \rangle$ 
3: else
4:   for each  $j \in A_m$  do
5:      $\langle V(j,s), a_j \rangle = Evaluate(j,s)$ 
6:    $j^{greedy} = argmax_j [V(j,s) + C(m,s,j)]$ 
7:   return  $\langle V(j^{greedy},s), a_{j^{greedy}} \rangle$ 

```

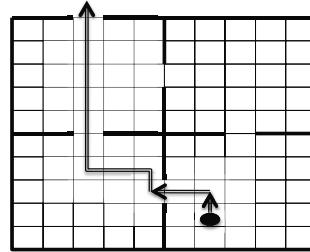
Algorithm 18. $Evaluate(m,s)$ [Dietterich (2000)]

Fig. 9.4 The completion function components of the decomposed value function for the agent following an optimal policy for the four-room problem in Figure 9.1. The agent is shown as a solid black oval at the starting state.

For the four-room task-hierarchy in Figure 9.2, the decomposed value of the agent's state has three terms determined by the two levels in the task-hierarchy plus a primitive action. With the agent located in the state shown in Figure 9.4 by a solid back oval, the optimal value function for this state is the cost of the shortest path out of the house. It is composed by adding the expected reward for taking the next primitive action to the North, completing the lower-level sub-task of leaving the room to the West, and completing the higher-level task of leaving the house.

MAXQ Learning and Execution

Given a designer specified task-hierarchy, Algorithm 19 performs the equivalent of on-line Q-Learning (Chapter 1) for completion functions for each of the subtask SMDPs in a task-hierarchy. If the action is primitive it learns the expected reward for that action in each state (Line 3). For abstract actions the completion function following the abstract action is updated (Line 12). The learning rate parameter α is gradually reduced to zero in the limit. Algorithm 19 is a simplified version of the

MAXQ algorithm. For an extended version of the Algorithm, one that accelerates learning and distinguishes goal from non-goal terminal states using *pseudo-rewards*, please see (Dietterich, 2000). Algorithm 20 initiates the MAXQ process that proceeds to learn and execute the task-hierarchy.

```

Require:  $V, C, A$ , learning rate  $\alpha$ 
1: if  $m$  is a primitive action then
2:   execute  $m$ , receive reward  $r$ , and observe the result state  $s'$ 
3:    $V(m,s) \leftarrow (1 - \alpha)V(m,s) + \alpha r$ 
4:   return 1
5: else
6:    $count = 0$ 
7:   while  $m$  has not terminated do
8:     choose an exploration action  $a$ 
9:      $N = MAXQ(a,s)$ 
10:    observe result state  $s'$ 
11:     $\langle V(j^{greedy}, s'), a_{j^{greedy}} \rangle \leftarrow Evaluate(m, s')$ 
12:     $C(m, s, a) \leftarrow (1 - \alpha)C(m, s, a) + \alpha \gamma^N V(j^{greedy}, s')$ 
13:     $count \leftarrow count + N$ 
14:     $s = s'$ 
15:   return  $count$ 
```

Algorithm 19. $MAXQ(m,s)$ [Dietterich (2000)]

```

Require: root node  $m_0$ , starting state  $s_0, V, C$ 
1: initialise  $V(m,s)$  and  $C(m,s,a)$  arbitrarily
2:  $MAXQ(m_0, s_0)$ 
```

Algorithm 20. Main Program MAXQ

HRL Applied to the Four-Room Task

We will now put all the above ideas together and show how we can learn and execute the four-room task in Figure 9.1 when the agent can start in any state. The designer of the task-hierarchy in Figure 9.5 has recognised several state abstraction opportunities. Recall that the state is described by the tuple $(room, position)$.

The agent can leave each room by one of four potential doorways to the North, East, South, or West, and we need to learn a separate navigation strategy for each. However, because the rooms are identical, the room variable is irrelevant for intra-room navigation.

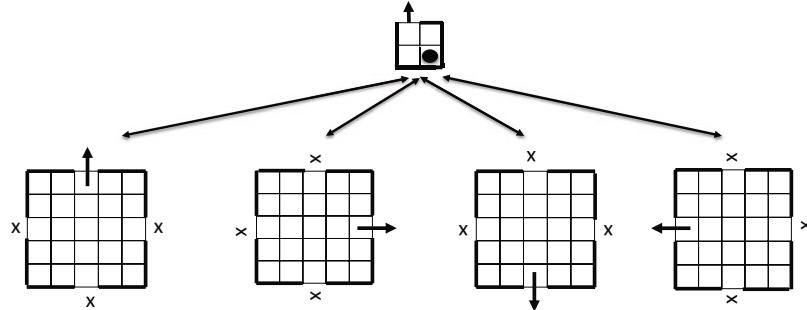


Fig. 9.5 A task task-hierarchy for the four-room task. The subtasks are room-leaving actions and can be used in any of the rooms. The parent-level root subtask has just four states representing the four rooms. “X” indicates non-goal terminal states.

We also notice that room-leaving abstract actions always terminate in one state. A room leaving abstract action is seen to “funnel” the agent through the doorway. It is for this reason that the position-in-room states can be abstracted away and the room state retained at the root level and only room leaving abstract actions are deployed at the root subtask. This means that instead of requiring 100 states for the root subtask we only require four, one for each room. Also, we only need to learn 16 (4 states \times 4 actions) completion functions, instead of 400.

To solve the problem using the task-hierarchy with Algorithm 19, a main program initialises expected primitive reward functions $V(\cdot, \cdot)$ and completion functions $C(\cdot, \cdot, \cdot)$ arbitrarily, and calls function $MAXQ$ at the root subtask in the task-hierarchy for starting state s_0 , i.e. $MAXQ(\text{root}, s_0)$. $MAXQ$ uses a Q -Learning like update rule to learn expected rewards for primitive actions and completion values for all subtasks.

With the values converged, α set to zero, and exploration turned off, $MAXQ$ (Algorithm 18) will execute a recursively optimal policy by searching for the shortest path to exit the four rooms. An example of such a path and its decomposed value function is shown in Figure 9.4 for one starting position.

9.4 Learning Structure

In most hierarchical reinforcement learning (HRL) applications the structure of the task-hierarchy or the partial program is provided as background knowledge by the designer. Automating the decomposition of a problem appears to be more difficult. It is desirable to have machines free designers from this task. Hierarchical structure requires knowledge of how a problem can best be decomposed and how to introduce state abstractions to balance complexity against loss of optimality.

Readers may be familiar with the mutilated checker-board problem showing that problem representation plays a large part in its solution (Gamow and Stern, 1958). In his seminal paper on six different representations for the missionaries and cannibals problem, Amarel (1968) demonstrated the possibility of making machine learning easier by discovering regularities and using them to formulating new representations. The choice of variables to represent states and actions in a reinforcement learning problem plays a large part in providing opportunities to decompose the problem.

Some researchers have tried to learn the hierarchical structure from the agent-environment interaction. Most approaches look for sub-goals or sub-tasks that try to partition the problem into near independent reusable sub-problems. Methods to automatically decompose problems include ones that look for sub-goal bottleneck or landmark states, and ones that find common behaviour trajectories or region policies.

Learning from the Bottom-Up

Automatic machine decomposition of complex tasks through interactions with the environment requires learning in stages. While a programmer can specify behaviour from the top down, learning seems to evolve from the bottom up.

Simon (1996) observed – “complex systems will evolve from simple systems much more rapidly if there are stable intermediate forms”. Clark and Thornton (1997) present a persuasive argument for modelling complex environments, namely, that it is necessary to proceed bottom-up and solve simple problems as intermediate representations. In their words,

... the underlying trick is always the same; to maximise the role of achieved representations, and thus minimise the space of subsequent search.

This is consistent with the principles advocated by Stone (1998) that include problem decomposition into multi-layers of abstraction, learning tasks from the lowest level to the highest in a hierarchy, where the output of learning from one layer feeds into the next layer. Utgoff and Stracuzzi (2002) point to the compression inherent in the progression of learning from simple to more complex tasks. They suggest a building block approach, designed to eliminate replication of knowledge structures. Agents are seen to advance their knowledge by moving their “frontier of receptivity” as they acquire new concepts by building on earlier ones from the bottom up. Their conclusion:

“Learning of complex structures can be guided successfully by assuming that local learning methods are limited to simple tasks, and that the resulting building blocks are available for subsequent learning”.

Some of the approaches use to learn abstract actions and structure include searching for common behaviour trajectories or common state region polices (Thrun and Schwartz, 1995; McGovern, 2002). Others look for bottleneck or landmark states

(Digney, 1998; McGovern, 2002; Menache et al, 2002). Şimşek and Barto (2004) use a relative novelty measure to identify sub-goal states. Interestingly Moore et al (1999) suggest that, for some navigation tasks, performance is insensitive to the position of landmarks and an (automatic) randomly-generated set of landmarks does not show widely varying results from ones more purposefully positioned.

9.4.1 HEXQ

We now describe one approach to automatic machine learning of hierarchic structure. The HEXQ (hierarchical exit Q function) approach is a series of algorithms motivated by MAXQ value-function decomposition and bottom-up structure learning. The algorithms rely on a finitely dimensioned (factored) base level state description. An underlying MDP is assumed but not known to the agent. The objective is to learn a task-hierarchy and find an optimal policy.

HEXQ constructs a hierarchy starting from base level states and primitive actions. It finds abstract actions in those parts of the state-space where some variables are irrelevant. It then recursively formulates reduced SMDPs at higher levels using the result-states from the “funnel” behaviour of the previously found abstract actions.

Learning Abstract Actions and the Task-Hierarchy

HEXQ searches for subspaces by exploring the transition and reward functions for the projected state space onto each state variable. Subspace states are included in the same block of a partition when transitions do not change the other variables and the transition and reward functions are independent of other variables. This is a stricter form of stochastic bisimulation homogeneity (Section 9.2.4), one where the context, in the guise of other than the projected variable, does not change. The associated state abstraction eliminates context variables from the subspace as they are irrelevant.

Whenever these condition are violated for a state transition, an *exit*, represented as a state-action pair, (s,a) , is created. If exits cannot be reached from initial subspace states, the subspace is split and extra exits created. Creating exits is the mechanism by which subgoals are automatically generated.

This process partitions the projected state space for each variable. Each block of the partition forms a set of subtasks, one for each exit. The subtask is a SMDP with the goal of terminating on exit execution. HEXQ subtasks are like options, except that the termination condition is defined by a state-action pair. The reward on termination is not a part of the subtask, but is counted at higher levels in the task-hierarchy. This represents a slight reformulation of the MAXQ value function decomposition, but one that unifies the definition of the Q function over the task-hierarchy and simplifies the recursive equations.

Early versions of HEXQ learn a monolithic hierarchy after ordering state variables by their frequency of change (Hengst, 2002). In more recent versions, state variables are tackled in parallel (Hengst, 2008). The projected state-space for each variable is partitioned into blocks as before. Parent level states are new variables formed by taking the cross-product of block identifiers from the child variables. This method of parallel decomposition of the factored state-space generates *sequential (or multi-tasking)* actions, that invoke multiple child subtasks, one at a time. Sequential actions create *partial-order* task-hierarchies that have the potential for greater state abstraction. (Hengst, 2008).

The Four-room HEXQ Decomposition

For the four-room task, HEXQ is provided with the factored state variable pairs (*room, position*). It starts by forming one module for each variable. The position module discovers one block with four exits. Exits are transitions that leave any room by one of the doorways. They are discovered automatically because it is only for these transitions that the room variable may change. The position module will formulate four room-leaving subtasks, one for each of the exits. The subtask policies can be learnt in parallel using standard off-policy Q -learning for SMDPs.

The room module learns a partition of the room states that consists of singleton state blocks, because executing a primitive action in any room may change the position variable value. For each block there are four exits. The block identifier is just the room state and is combined with the block identifier from the position module to form a new state variable for the parent module.

In this case the room module does not add any value in the form of state abstraction and could be eliminated with the room variable passed directly to the parent module as part of the input. Either way the parent module now represents abstract room states and invokes room-leaving abstract actions to achieve its goal. The machine generated task-hierarchy is similar to that shown in Figure 9.5.

Interestingly, if the four-room state is defined using coordinates (x,y) , where x and y range in value from 0 to 9, instead of the more descriptive (*room, position*), HEXQ will nevertheless create a higher level variable representing four rooms, but one that was not supplied by the designer. Each of the x and y variables are partitioned by HEXQ into two blocks representing the space inside each room. The cross-product of the block identifiers creates a higher-level variable representing the four rooms. Sequential abstract actions are generated to move East-West and North-South to leave rooms. In this decomposition the subspaces have a different meaning. They model individual rooms instead of a generic room (Hengst, 2008).

Summary of HEXQ

HEXQ automatically builds task-hierarchies from interactions with the environment assuming an underlying finite state factored MDP. It employs both variable

elimination and funnel type state abstractions to construct a compact representation (Section 9.2.4). As one subtask is learnt for each exit, HEXQ solutions are hierarchically optimal (Section 9.2.6). HEXQ can use hierarchical greedy execution to try to improve on the hierarchically optimal solution, and the HEXQ policy converges to the globally optimal result for task-hierarchies where only the root-subtask has stochastic transition or reward functions. The introduction of parallel decomposition of variables and sequential actions allows HEXQ to create abstract state variables not supplied by the designer. The decompositions is guaranteed to be “safe” in the sense that the decomposed value function of any policy over the task-hierarchy is equivalent to the value function when that policy is executed with the original MDP.

9.5 Related Work and Ongoing Research

Hierarchical reinforcement learning (HRL) is not an isolated subfield. In combination with topics covered in other chapters in this book it has generated several contributions and presents ongoing research challenges. We briefly list some of this work.

Our above treatment of HRL assumes discrete time, but SMDPs are defined more generally with continuous-time temporally extended actions (Puterman, 1994). Ghavamzadeh and Mahadevan (2001) extend MAXQ to continuous-time models for both the discounted reward and average reward optimality models introduced in Chapter 1. Many real-world problems involve continuous variables. Ghavamzadeh and Mahadevan (2003) use a hierarchical policy gradient approach to tackle reinforcement learning problems with continuous state and action spaces. Jong and Stone (2009) combine function approximation and optimistic exploration to allow MAXQ to cope with large and even infinite state spaces.

We have seen how the elimination of irrelevant variables can lead to the implicit transfer of skills between different contexts. Hierarchical decomposition can play a significant role in *transfer learning* where experience gained in solving one task can help in solving a different, but related task. Konidaris and Barto (2007) build portable options that distinguishing between *agent-space* and *problem-space*. The former agent-centric representation is closely related to the notion of deictic representations (Agre and Chapman, 1987). Marthi et al (2007) use sample trajectories to learn MAXQ style hierarchies where the objective function measures how much planning or reinforcement learning is likely to be simplified on future similar environments by using this hierarchy. Mehta et al (2008a) transfer the value function between different SMDPs and find that transfer is especially effective in the hierarchical setting. Mehta et al (2008b) discover MAXQ task hierarchies by applying dynamic Bayesian network models to a successful trajectory from a source reinforcement learning task. An interesting conclusion is that transferring the knowledge about the structure of the task-hierarchy may be more important than transferring the value function. The latter can even hinder the learning on the new task as it has to unlearn the transferred policy. Taylor and Stone (2009) survey transfer learning in

the reinforcement setting including abstract actions, state abstraction and hierarchical approaches. For effective knowledge transfer the two environments need to be “close enough”. Castro and Precup (2010) show that transferring abstract actions is more successful than just primitive actions using a variant of the bisimulation metric.

State abstraction for MAXQ like task-hierarchies is hindered by discounted reward optimality models because the rewards after taking an abstract action are no longer independent of the time to complete the abstract action. Hengst (2007) has developed a method for decomposing discounted value functions over the task-hierarchy by concurrently decomposing multi-time models (Precup and Sutton, 1997). The twin recursive decomposition functions restore state-abstraction opportunities and allow problems with continuing subtasks in the task-hierarchy to be included in the class of problems that can be tackled by HRL.

Much of the literature in reinforcement learning involves one-dimensional actions. However, in many domains we wish to control several action variables simultaneously. These situations arise, for example, when coordinating teams of robots, or when individual robots have multiple degrees of articulation. The challenge is to decompose factored actions over a task-hierarchy, particularly if there is a chance that the abstract actions will interact. Rohanimanesh and Mahadevan (2001) use the options framework to show how actions can be parallelized with a SMPD formalism. Fitch et al (2005) demonstrate, using a two-taxi task, concurrent action task-hierarchies and state abstraction to scale problems involving concurrent actions. Concurrent actions require a subtask termination scheme (Rohanimanesh and Mahadevan, 2005) and the attribution of subtask rewards. Marthi et al (2005) extend partial programs (Section 9.2.3) to the concurrent action case using multi-threaded concurrent-ALisp.

When the state is not fully observable, or the observations are noisy the MDP becomes a *Partially Observable Markov Decision Problem* or POMDP (Chapter 12). Wiering and Schmidhuber (1997) decompose a POMDP into sequences of simpler subtasks, Hernandez and Mahadevan (2000) solve partially observable sequential decision tasks by propagating reward across long decision sequences using a memory-based SMDP. Pineau and Thrun (2002) present an algorithm for planning in structured POMDPs using an action based decomposition to partition a complex problem into a hierarchy of smaller subproblems. Theocharous and Kaelbling (2004) derive a hierarchical partially observable Markov decision problem (HPOMDP) from hierarchical hidden Markov models extending previous work to include multiple entry and exit states to represent the spatial borders of the sub-space.

New structure learning techniques continue to be developed. The original HEXQ decomposition uses a simple heuristic to determine an ordering over the state variables for the decomposition. Jonsson and Barto (2006) propose a Bayesian network model causal graph based approach – Variable Influence Structure Analysis (VISA) – that relates the way variables influence each other to construct the task-hierarchy. Unlike HEXQ this algorithm combines variables that influence each other and ignores lower-level activity. Bakker and Schmidhuber (2004)’s HASSLE

algorithm discovers subgoals by learning how to transition between abstract states. In the process subgoal abstract actions are generalised to be reused or specialised to work in different parts of the state-space or to reach different goals. HASSLE is extended with function approximation by Moerman (2009). Strehl et al (2007) learn dynamic Bayesian network (DBN) structures as part of the reinforcement learning process using a factored state representation.

Mugan and Kuipers (2009) present a method for learning a hierarchy of actions in a continuous environment by learning a qualitative representation of the continuous environment and then find actions to reach qualitative states. Neumann et al (2009) learn to parameterize and order a set of motion templates to form abstract actions (options) in continuous time. Konidaris and Barto (2009) introduce a skill discovery method for reinforcement learning in continuous domains that constructs chains of skills leading to an end-of-task reward. The method is further developed to build skill trees faster from a set of sample solution trajectories Konidaris et al (2010).

Osentoski and Mahadevan (2010) extend automatic basis function construction to HRL. The approach is based on hierarchical spectral analysis of graphs induced on an SMDP's state space from sample trajectories. Mahadevan (2010) provides a brief review of more recent progress in general representation discovery. The review includes temporal and homomorphic state abstractions, with the latter generalised to representing value functions abstractly in terms of a basis functions.

9.6 Summary

Hierarchical Reinforcement Learning (HRL) decomposes a reinforcement learning problem into a hierarchy of sub-tasks such that higher-level parent-tasks invoke lower-level child tasks as if they were primitive actions. A decomposition may have multiple levels of hierarchy. Some or all of the sub-problems can themselves be reinforcement learning problems. When a parent-task is formulated as a reinforcement learning problem it is commonly formalised as a semi-Markov Decision Problem because its actions are child-tasks that persist for an extended period of time. The advantage of hierarchical decomposition is a reduction in computational complexity if the overall problem can be represented more compactly and reusable sub-tasks learned or provided independently. There is usually a trade-off between the reduction in problem complexity through decomposition and how close the solution of the decomposed problem is to optimal.

References

- Agre, P.E., Chapman, D.: Pengi: an implementation of a theory of activity. In: Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI 1987, vol. 1, pp. 268–272. AAAI Press (1987)

- Amarel, S.: On representations of problems of reasoning about actions. In: Michie, D. (ed.) *Machine Intelligence*, vol. 3, pp. 131–171. Edinburgh at the University Press, Edinburgh (1968)
- Andre, D., Russell, S.J.: Programmable reinforcement learning agents. In: Leen, T.K., Dietterich, T.G., Tresp, V. (eds.) *NIPS*, pp. 1019–1025. MIT Press (2000)
- Andre, D., Russell, S.J.: State abstraction for programmable reinforcement learning agents. In: Dechter, R., Kearns, M., Sutton, R.S. (eds.) *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 119–125. AAAI Press (2002)
- Ashby, R.: *Design for a Brain: The Origin of Adaptive Behaviour*. Chapman & Hall, London (1952)
- Ashby, R.: *Introduction to Cybernetics*. Chapman & Hall, London (1956)
- Bakker, B., Schmidhuber, J.: Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In: *Proceedings of the 8-th Conference on Intelligent Autonomous Systems*, IAS-8, pp. 438–445 (2004)
- Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Special Issue on Reinforcement Learning, *Discrete Event Systems Journal* 13, 41–77 (2003)
- Bellman, R.: *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton (1961)
- Boutilier, C., Dearden, R., Goldszmidt, M.: Exploiting structure in policy construction. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, vol. 2, pp. 1104–1111. Morgan Kaufmann Publishers Inc., San Francisco (1995)
- Boutilier, C., Reiter, R., Soutchanski, M., Thrun, S.: Decision-theoretic, high-level agent programming in the situation calculus. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pp. 355–362. AAAI Press (2000)
- Brooks, R.A.: Elephants don't play chess. *Robotics and Autonomous Systems* 6, 3–15 (1990)
- Castro, P.S., Precup, D.: Using bisimulation for policy transfer in mdps. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS 2010, vol. 1, pp. 1399–1400. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2010)
- Clark, A., Thornton, C.: Trading spaces: Computation, representation, and the limits of uninformed learning. *Behavioral and Brain Sciences* 20(1), 57–66 (1997)
- Dayan, P., Hinton, G.E.: Feudal reinforcement learning. In: *Advances in Neural Information Processing Systems (NIPS)*, vol. 5 (1992)
- Dean, T., Givan, R.: Model minimization in Markov decision processes. In: *AAAI/IAAI*, pp. 106–111 (1997)
- Dean, T., Lin, S.H.: Decomposition techniques for planning in stochastic domains. Tech. Rep. CS-95-10, Department of Computer Science Brown University (1995)
- Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13, 227–303 (2000)
- Digney, B.L.: Learning hierarchical control structures for multiple tasks and changing environments. From Animals to Animats 5: *Proceedings of the Fifth International Conference on Simulation of Adaptive Behaviour* SAB (1998)
- Ferrein, A., Lakemeyer, G.: Logic-based robot control in highly dynamic domains. *Robot Auton. Syst.* 56(11), 980–991 (2008)
- Fitch, R., Hengst, B., Šuc, D., Calbert, G., Scholz, J.: Structural Abstraction Experiments in Reinforcement Learning. In: Zhang, S., Jarvis, R.A. (eds.) *AI 2005. LNCS (LNAI)*, vol. 3809, pp. 164–175. Springer, Heidelberg (2005)

- Forestier, J., Varaiya, P.: Multilayer control of large Markov chains. *IEEE Transactions Automatic Control* 23, 298–304 (1978)
- Gamow, G., Stern, M.: *Puzzle-math*. Viking Press (1958)
- Ghavamzadeh, M., Mahadevan, S.: Continuous-time hierachial reinforcement learning. In: Proc. 18th International Conf. on Machine Learning, pp. 186–193. Morgan Kaufmann, San Francisco (2001)
- Ghavamzadeh, M., Mahadevan, S.: Hierarchical policy gradient algorithms. In: Marine Environments, pp. 226–233. AAAI Press (2003)
- Hauskrecht, M., Meuleau, N., Kaelbling, L.P., Dean, T., Boutilier, C.: Hierarchical solution of Markov decision processes using macro-actions. In: Fourteenth Annual Conference on Uncertainty in Artificial Intelligence, pp. 220–229 (1998)
- Hengst, B.: Discovering hierarchy in reinforcement learning with HEXQ. In: Sammut, C., Hoffmann, A. (eds.) *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 243–250. Morgan Kaufmann (2002)
- Hengst, B.: Model Approximation for HEXQ Hierarchical Reinforcement Learning. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 144–155. Springer, Heidelberg (2004)
- Hengst, B.: Safe State Abstraction and Reusable Continuing Subtasks in Hierarchical Reinforcement Learning. In: Orgun, M.A., Thornton, J. (eds.) AI 2007. LNCS (LNAI), vol. 4830, pp. 58–67. Springer, Heidelberg (2007)
- Hengst, B.: Partial Order Hierarchical Reinforcement Learning. In: Wobcke, W., Zhang, M. (eds.) AI 2008. LNCS (LNAI), vol. 5360, pp. 138–149. Springer, Heidelberg (2008)
- Hernandez, N., Mahadevan, S.: Hierarchical memory-based reinforcement learning. In: Fifteenth International Conference on Neural Information Processing Systems, Denver (2000)
- Hutter, M.: Universal algorithmic intelligence: A mathematical top→down approach. In: Artificial General Intelligence, pp. 227–290. Springer, Berlin (2007)
- Jong, N.K., Stone, P.: Compositional models for reinforcement learning. In: The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (2009)
- Jonsson, A., Barto, A.G.: Causal graph based decomposition of factored mdps. *Journal of Machine Learning* 7, 2259–2301 (2006)
- Kaelbling, L.P.: Hierarchical learning in stochastic domains: Preliminary results. In: Proceedings of the Tenth International Conference Machine Learning, pp. 167–173. Morgan Kaufmann, San Mateo (1993)
- Konidaris, G., Barto, A.G.: Building portable options: skill transfer in reinforcement learning. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence, pp. 895–900. Morgan Kaufmann Publishers Inc., San Francisco (2007)
- Konidaris, G., Barto, A.G.: Skill discovery in continuous reinforcement learning domains using skill chaining. In: Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C.K.I., Culotta, A. (eds.) *Advances in Neural Information Processing Systems*, vol. 22, pp. 1015–1023 (2009)
- Konidaris, G., Kuindersma, S., Barto, A.G., Grupen, R.: Constructing skill trees for reinforcement learning agents from demonstration trajectories. In: *Advances in Neural Information Processing Systems NIPS*, vol. 23 (2010)
- Korf, R.E.: *Learning to Solve Problems by Searching for Macro-Operators*. Pitman Publishing Inc., Boston (1985)
- Levesque, H., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.: Golog: A logic programming language for dynamic domains. *Journal of Logic Programming* 31, 59–84 (1997)

- Mahadevan, S.: Representation discovery in sequential descision making. In: 24th Conference on Artificial Intelligence (AAAI), Atlanta, July 11-15 (2010)
- Marthi, B., Russell, S., Latham, D., Guestrin, C.: Concurrent hierarchical reinforcement learning. In: Proc. IJCAI 2005 Edinburgh, Scotland (2005)
- Marthi, B., Kaelbling, L., Lozano-Perez, T.: Learning hierarchical structure in policies. In: NIPS 2007 Workshop on Hierarchical Organization of Behavior (2007)
- McGovern, A.: Autonomous Discovery of Abstractions Through Interaction with an Environment. In: Koenig, S., Holte, R.C. (eds.) SARA 2002. LNCS (LNAI), vol. 2371, pp. 338–339. Springer, Heidelberg (2002)
- Mehta, N., Natarajan, S., Tadepalli, P., Fern, A.: Transfer in variable-reward hierarchical reinforcement learning. *Mach. Learn.* 73, 289–312 (2008a), doi:10.1007/s10994-008-5061-y
- Mehta, N., Ray, S., Tadepalli, P., Dietterich, T.: Automatic discovery and transfer of maxq hierarchies. In: Proceedings of the 25th International Conference on Machine Learning, ICML 2008, pp. 648–655. ACM, New York (2008b)
- Menache, I., Mannor, S., Shimkin, N.: Q-Cut - Dynamic Discovery of Sub-goals in Reinforcement Learning. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) ECML 2002. LNCS (LNAI), vol. 2430, pp. 295–305. Springer, Heidelberg (2002)
- Moerman, W.: Hierarchical reinforcement learning: Assignment of behaviours to subpolicies by self-organization. PhD thesis, Cognitive Artificial Intelligence, Utrecht University (2009)
- Moore, A., Baird, L., Kaelbling, L.P.: Multi-value-functions: Efficient automatic action hierarchies for multiple goal mdps. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 1316–1323. Morgan Kaufmann, San Francisco (1999)
- Mugan, J., Kuipers, B.: Autonomously learning an action hierarchy using a learned qualitative state representation. In: Proceedings of the 21st International Jont Conference on Artifical Intelligence, pp. 1175–1180. Morgan Kaufmann Publishers Inc., San Francisco (2009)
- Neumann, G., Maass, W., Peters, J.: Learning complex motions by sequencing simpler motion templates. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, pp. 753–760. ACM, New York (2009)
- Nilsson, N.J.: Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research* 1, 139–158 (1994)
- Osentoski, S., Mahadevan, S.: Basis function construction for hierarchical reinforcement learning. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2010, vol. 1, pp. 747–754. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2010)
- Parr, R., Russell, S.J.: Reinforcement learning with hierarchies of machines. In: NIPS (1997)
- Parr, R.E.: Hierarchical control and learning for Markov decision processes. PhD thesis, University of California at Berkeley (1998)
- Pineau, J., Thrun, S.: An integrated approach to hierarchy and abstraction for pomdps. CMU Technical Report: CMU-RI-TR-02-21 (2002)
- Polya, G.: How to Solve It: A New Aspect of Mathematical Model. Princeton University Press (1945)
- Precup, D., Sutton, R.S.: Multi-time models for temporally abstract planning. In: Advances in Neural Information Processing Systems, vol. 10, pp. 1050–1056. MIT Press (1997)
- Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Whiley & Sons, Inc., New York (1994)
- Ravindran, B., Barto, A.G.: SMDP homomorphisms: An algebraic approach to abstraction in semi Markov decision processes. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI 2003 (2003)

- Rohanimanesh, K., Mahadevan, S.: Decision-theoretic planning with concurrent temporally extended actions. In: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, pp. 472–479. Morgan Kaufmann Publishers Inc., San Francisco (2001)
- Rohanimanesh, K., Mahadevan, S.: Coarticulation: an approach for generating concurrent plans in Markov decision processes. In: ICML 2005: Proceedings of the 22nd international conference on Machine learning, pp. 720–727. ACM Press, New York (2005)
- Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River (1995)
- Ryan, M.R.K.: Hierarchical Decision Making. In: Handbook of Learning and Approximate Dynamic Programming. IEEE Press Series on Computational Intelligence. Wiley-IEEE Press (2004)
- Reid, M.D., Ryan, M.: Using ILP to Improve Planning in Hierarchical Reinforcement Learning. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 174–190. Springer, Heidelberg (2000)
- Si, J., Barto, A.G., Powell, W.B., Wunsch, D.: Handbook of Learning and Approximate Dynamic Programming. IEEE Press Series on Computational Intelligence. Wiley-IEEE Press (2004)
- Simon, H.A.: The Sciences of the Artificial, 3rd edn. MIT Press, Cambridge (1996)
- Şimşek, O., Barto, A.G.: Using relative novelty to identify useful temporal abstractions in reinforcement learning. In: Proceedings of the Twenty-First International Conference on Machine Learning, ICML 2004 (2004)
- Singh, S.: Reinforcement learning with a hierarchy of abstract models. In: Proceedings of the Tenth National Conference on Artificial Intelligence (1992)
- Stone, P.: Layered learning in multi-agent systems. PhD, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA (1998)
- Strehl, A.L., Diuk, C., Littman, M.L.: Efficient structure learning in factored-state mdps. In: Proceedings of the 22nd National Conference on Artificial Intelligence, vol. 1, pp. 645–650. AAAI Press (2007)
- Sutton, R.S., Precup, D., Singh, S.P.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112(1-2), 181–211 (1999)
- Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research 10(1), 1633–1685 (2009)
- Theocharous, G., Kaelbling, L.P.: Approximate planning in POMDPs with macro-actions. In: Advances in Neural Information Processing Systems 16 (NIPS-2003) (2004) (to appear)
- Thrun, S., Schwartz, A.: Finding structure in reinforcement learning. In: Tesauro, G., Touretzky, D., Leen, T. (eds.) Advances in Neural Information Processing Systems (NIPS), vol. 7. MIT Press, Cambridge (1995)
- Utgoff, P.E., Stracuzzi, D.J.: Many-layered learning. In: Neural Computation. MIT Press Journals (2002)
- Watkins CJCH, Learning from delayed rewards. PhD thesis, King's College (1989)
- Wiering, M., Schmidhuber, J.: HQ-learning. Adaptive Behavior 6, 219–246 (1997)

Chapter 10

Evolutionary Computation for Reinforcement Learning

Shimon Whiteson

Abstract. Algorithms for evolutionary computation, which simulate the process of natural selection to solve optimization problems, are an effective tool for discovering high-performing reinforcement-learning policies. Because they can automatically find good representations, handle continuous action spaces, and cope with partial observability, evolutionary reinforcement-learning approaches have a strong empirical track record, sometimes significantly outperforming temporal-difference methods. This chapter surveys research on the application of evolutionary computation to reinforcement learning, overviewing methods for evolving neural-network topologies and weights, hybrid methods that also use temporal-difference methods, coevolutionary methods for multi-agent settings, generative and developmental systems, and methods for on-line evolutionary reinforcement learning.

10.1 Introduction

Algorithms for *evolutionary computation*, sometimes known as *genetic algorithms* (Holland, 1975; Goldberg, 1989), are optimization methods that simulate the process of natural selection to find highly fit solutions to a given problem. Typically the problem assumes as input a *fitness function* $f : C \rightarrow \mathbb{R}$ that maps C , the set of all candidate solutions, to a real-valued measure of fitness. The goal of an optimization method is to find $c^* = \arg \max_c f(c)$, the fittest solution. In some cases, the fitness function may be stochastic, in which case $f(c)$ can be thought of as a random variable and $c^* = \arg \max_c E[f(c)]$.

Evolutionary methods search for c^* by repeatedly selecting and reproducing a population of candidate solutions. The initial population is typically chosen randomly, after which each member of the population is evaluated using f and the

Shimon Whiteson
Informatics Institute, University of Amsterdam
e-mail: s.a.whiteson@uva.nl

best performing ones are selected as the basis for a new population. This new population is formed via reproduction, in which the selected policies are mated (i.e., components of two different solutions are combined) and mutated (i.e., the parameter values of one solution are stochastically altered). This process repeats over many iterations, until a sufficiently fit solution has been found or the available computational resources have been exhausted.

There is an enormous number of variations on this approach, such as multi-objective methods (Deb, 2001; Coello et al, 2007) diversifying algorithms (Holland, 1975; Goldberg and Richardson, 1987; Mahfoud, 1995; Potter and De Jong, 1995; Darwen and Yao, 1996) and distribution-based methods (Larranaga and Lozano, 2002; Hansen et al, 2003; Rubinstein and Kroese, 2004). However, the basic approach is extremely general and can in principle be applied to all optimization problems for which f can be specified.

Included among these optimization problems are reinforcement-learning tasks (Moriarty et al, 1999). In this case, C corresponds to the set of possible policies, e.g., mappings from S to A , and $f(c)$ is the average cumulative reward obtained while using such a policy in a series of Monte Carlo trials in the task. In other words, in an evolutionary approach to reinforcement learning, the algorithm directly searches the space of policies for one that maximizes the expected cumulative reward.

Like many other policy-search methods, this approach reasons only about the value of entire policies, without constructing value estimates for particular state-action pairs, as temporal-difference methods do. The holistic nature of this approach is sometimes criticized. For example, Sutton and Barto write:

Evolutionary methods do not use the fact that the policy they are searching for is a function from states to actions; they do not notice which states an individual passes through during its lifetime, or which actions it selects. In some cases this information can be misleading (e.g., when states are misperceived) but more often it should enable more efficient search (Sutton and Barto, 1998, p. 9).

These facts can put evolutionary methods at a theoretical disadvantage. For example, in some circumstances, dynamic programming methods are guaranteed to find an optimal policy in time polynomial in the number of states and actions (Littman et al, 1995). By contrast, evolutionary methods, in the worst case, must iterate over an exponential number of candidate policies before finding the best one. Empirical results have also shown that evolutionary methods sometimes require more episodes than temporal-difference methods to find a good policy, especially in highly stochastic tasks in which many Monte Carlo simulations are necessary to achieve a reliable estimate of the fitness of each candidate policy (Runarsson and Lucas, 2005; Lucas and Runarsson, 2006; Lucas and Togelius, 2007; Whiteson et al, 2010b).

However, despite these limitations, evolutionary computation remains a popular tool for solving reinforcement-learning problems and boasts a wide range of empirical successes, sometimes substantially outperforming temporal-difference methods (Whitley et al, 1993; Moriarty and Miikkulainen, 1996; Stanley and Miikkulainen, 2002; Gomez et al, 2008; Whiteson et al, 2010b). There are three main reasons why.

First, evolutionary methods can cope well with partial observability. While evolutionary methods do not exploit the relationship between subsequent states that an

agent visits, this can be advantageous when the agent is unsure about its state. Since temporal-difference methods rely explicitly on the Markov property, their value estimates can diverge when it fails to hold, with potentially catastrophic consequences for the performance of the greedy policy. In contrast, evolutionary methods do not rely on the Markov property and will always select the best policies they can find for the given task. Severe partial observability may place a ceiling on the performance of such policies, but optimization within the given policy space proceeds normally (Moriarty et al, 1999). In addition, representations that use memory to reduce partial observability, such as recurrent neural networks, can be optimized in a natural way with evolutionary methods (Gomez and Miikkulainen, 1999; Stanley and Miikkulainen, 2002; Gomez and Schmidhuber, 2005a,b).

Second, evolutionary methods can make it easier to find suitable representations for the agent's solution. Since policies need only specify an action for each state, instead of the value of each state-action pair, they can be simpler to represent. In addition, it is possible to simultaneously evolve a suitable policy representation (see Sections 10.3 and 10.4.2). Furthermore, since it is not necessary to perform learning updates on a given candidate solution, it is possible to use more elaborate representations, such as those employed by *generative and developmental systems* (GDS) (see Section 10.6).

Third, evolutionary methods provide a simple way to solve problems with large or continuous action spaces. Many temporal-difference methods are ill-suited to such tasks because they require iterating over the action space in each state in order to identify the maximizing action. In contrast, evolutionary methods need only evolve policies that directly map states to actions. Of course, actor-critic methods (Doya, 2000; Peters and Schaal, 2008) and other techniques (Gaskett et al, 1999; Millán et al, 2002; van Hasselt and Wiering, 2007) can also be used to make temporal-difference methods suitable for continuous action spaces. Nonetheless, evolutionary methods provide a simple, effective way to address such difficulties.

Of course, none of these arguments are unique to evolutionary methods, but apply in principle to other policy-search methods too. However, evolutionary methods have proven a particularly popular way to search policy space and, consequently, there is a rich collection of algorithms and results for the reinforcement-learning setting. Furthermore, as modern methods, such as distribution-based approaches, depart further from the original genetic algorithms, their resemblance to the process of natural selection has decreased. Thus, the distinction between evolutionary methods and other policy search approaches has become fuzzier and less important.

This chapter provides an introduction to and overview of evolutionary methods for reinforcement learning. The vastness of the field makes it infeasible to address all the important developments and results. In the interest of clarity and brevity, this chapter focuses heavily on *neuroevolution* (Yao, 1999), in which evolutionary methods are used to evolve *neural networks* (Haykin, 1994), e.g., to represent policies. While evolutionary reinforcement learning is by no means limited to neural-network representations, neuroevolutionary approaches are by far the most common. Furthermore, since neural networks are a popular and well-studied representation in general, they are a suitable object of focus for this chapter.

The rest of this chapter is organized as follows. By way of introduction, Section 10.2 describes a simple neuroevolutionary algorithm for reinforcement learning. Section 10.3 considers *topology- and weight-evolving artificial neural networks* (TWEANNs), including the popular NEAT method, that automatically discover their own internal representations. Section 10.4 considers hybrid approaches, such as *evolutionary function approximation* and *learning classifier systems*, that integrate evolution with temporal-difference methods. Section 10.5 discusses *coevolution*, in which multiple competing and/or cooperating policies are evolved simultaneously. Section 10.6 describes *generative and developmental systems* (GDSs) such as HyperNEAT, which rely on *indirect encodings*: more complex representations in which the agent's policy must be constructed or grown from the evolved parameter values. Section 10.7 discusses on-line methods that strive to maximize reward during evolution instead of merely discovering a good policy quickly.

10.2 Neuroevolution

Neural networks (Haykin, 1994) are an extremely general-purpose way of representing complex functions. Because of their concision, they are a popular representation for reinforcement-learning policies, not only for evolutionary computation, but also for other policy-search algorithms and for temporal-difference methods. This section introduces the basics of neuroevolutionary approaches to reinforcement learning, in which evolutionary methods are used to optimize neural-network policies.

Figure 10.1 illustrates the basic steps of a neuroevolutionary algorithm (Yao, 1999). In each generation, each network in the population is evaluated in the task. Next, the best performing are selected, e.g., via rank-based selection, roulette wheel selection, or tournament selection (Goldberg and Deb, 1991). The selected networks are bred via crossover and mutation and reproduced (Sywerda, 1989; De Jong and Spears, 1991)) to form a new population and the process repeats.

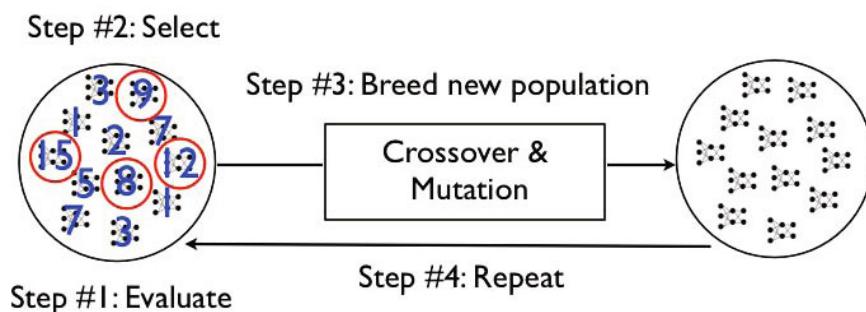


Fig. 10.1 The basic steps of neuroevolution

In a reinforcement-learning setting, each input node in a network typically corresponds to a state feature, such that the value of the inputs together describe the agent’s state. There are many ways to represent actions. If the actions can also be described with features, as is common with continuous action spaces, then each output node can correspond to an action feature. In this case, the value of the outputs together describe the action to be selected for the given state. When the number of actions is small and discrete, a separate network can be used for each action, as is common in value-function approximation. In this case, each network has only one output node. The policy’s action for a given state then corresponds to the network that produces the largest output for that state.

In this chapter, we focus on a variation of this approach called *action-selection* networks. As before, we assume the set of actions is small and discrete. However, only one network is used to represent the policy. This network has one output node for each action. The policy’s action for a given state then corresponds to the node that produces the largest output for that state.

Algorithm 21 contains a high-level description of a simple neuroevolutionary method that evolves action-selection networks for an episodic reinforcement-learning problem. It begins by creating a population of random networks (line 4). In each generation, it repeatedly iterates over the current population (lines 6–7). During each step of a given episode, the agent takes whatever action corresponds to the output with the highest activation (lines 10–12). Note that s' and a' are the current state and action while s and a are the previous state and action. Neuroevolution maintains a running total of the reward accrued by the network during its evaluation (line 13). Each generation ends after e episodes, at which point each network’s average fitness is $N.\text{fitness}/N.\text{episodes}$. In stochastic domains, e typically must be much larger than $|P|$ to ensure accurate fitness estimates for each network. Neuroevolution creates a new population by repeatedly calling the BREED-NET function (line 18), which generates a new network from highly fit parents.

Note that, while the action selection described in lines 10–12 resembles greedy action selection from a value function, the network should not be interpreted as a value function.¹ Evolution does not search for the networks that best approximate the optimal value function. Instead, it searches for networks representing high performing policies. To perform well, a network need only generate more output for the optimal action than for other actions. Unlike with value functions, the scale of the outputs can be arbitrary, as well as the relative outputs of the non-selected actions.

The neural network employed by Algorithm 21 could be a simple *feed-forward* network or a more complex *recurrent* network. Recurrent neural networks can contain cycles (e.g., the output emerging from an output node can be fed into an input node). Consequently, such networks can contain internal state. In a reinforcement-learning context, this internal state can record aspects of the agent’s observation history, which can help it cope with partial observability (Wieland, 1991; Gomez and Miikkulainen, 1999; Moriarty et al., 1999; Stanley and Miikkulainen, 2002; Igel, 2003; Gomez and Schmidhuber, 2005a,b).

¹ This does not apply to the hybrid methods discussed in Section 10.4.

```

1: //  $S$ : set of all states,  $A$ : set of all actions,  $p$ : population size
2: //  $g$ : number of generations,  $e$ : episodes per generation
3:
4:  $P \leftarrow \text{INIT-POPULATION}(S, A, p)$  // create new population  $P$  with random networks
5: for  $i \leftarrow 1$  to  $g$  do
6:   for  $j \leftarrow 1$  to  $e$  do
7:      $N, s, s' \leftarrow P[j \% p], \text{null}, \text{INIT-STATE}(S)$  // select next network
8:     repeat
9:        $Q \leftarrow \text{EVAL-NET}(N, s')$  // evaluate selected network on current state
10:       $a' \leftarrow \text{argmax}_i Q[i]$  // select action with highest activation
11:       $s, a \leftarrow s', a'$ 
12:       $r, s' \leftarrow \text{TAKE-ACTION}(a')$  // take action and transition to new state
13:       $N.\text{fitness} \leftarrow N.\text{fitness} + r$  // update total reward accrued by  $N$ 
14:    until TERMINAL-STATE?( $s$ )
15:     $N.\text{episodes} \leftarrow N.\text{episodes} + 1$  // update total number of episodes for  $N$ 
16:     $P' \leftarrow \text{new array of size } p$  // new array will store next generation
17:    for  $j \leftarrow 1$  to  $p$  do
18:       $P'[j] \leftarrow \text{BREED-NET}(P)$  // make a new network based on fit parents in  $P$ 
19:     $P \leftarrow P'$ 

```

Algorithm 21. NEUROEVOLUTION(S, A, p, g, e)

While Algorithm 21 uses a traditional genetic algorithm to optimize neural-network weights, many variations are also possible. For example, *estimation of distribution algorithms* (EDAs) (Larrañaga and Lozano, 2002; Hansen et al, 2003; Rubinstein and Kroese, 2004) can be used instead. EDAs, also called *probabilistic model-building genetic algorithms* (PMBGAs), do not explicitly maintain a population of candidate solutions. Instead, they maintain a distribution over solutions. In each generation, candidate solutions are sampled from this distribution and evaluated. A subset is then selected and used to update the distribution using *density estimation* techniques, unsupervised learning techniques for approximating the distribution from which a set of samples was drawn.

One of the most popular and effective EDAs is the *covariance matrix adaptation evolution strategy* (CMA-ES) (Hansen et al, 2003), a variable-metric EDA in which the distribution is a multivariate Gaussian whose covariance matrix adapts over time. When used to optimize neural networks, the resulting method, called CMA-NeuroES, has proven effective on a wide range of reinforcement-learning tasks (Igel, 2003; Heidrich-Meisner and Igel, 2008, 2009a,b,c).

10.3 TWEANNs

In its simplest form, Algorithm 21 evolves only neural networks with fixed representations. In such a setup, all the networks in a particular evolutionary run have

the same *topology*, i.e., both the number of hidden nodes and the set of edges connecting the nodes are fixed. The networks differ only with respect to the weights of these edges, which are optimized by evolution. The use of fixed representations is by no means unique to neuroevolution. In fact, though methods exist for automatically discovering good representations for value-functions (Mahadevan and Maggini, 2007; Parr et al, 2007) temporal-difference methods typically also use fixed representations for function approximation.

Nonetheless, reliance on fixed representations is a significant limitation. The primary reason is that it requires the user of the algorithm to correctly specify a good representation in advance. Clearly, choosing too simple a representation will doom evolution to poor performance, since describing high quality solutions becomes impossible. However, choosing too complex a representation can be just as harmful. While such a representation can still describe good solutions, finding them may become infeasible. Since each weight in the network corresponds to a dimension of the search space, a representation with too many edges can lead to an intractable search problem.

In most tasks, the user is not able to correctly guess the right representation. Even in cases where the user possesses great domain expertise, deducing the right representation from this expertise is often not possible. Typically, finding a good representation becomes a process of trial and error. However, repeatedly running evolution until a suitable representation is found greatly increases computational costs. Furthermore, in on-line tasks (see Section 10.7) it also increases the real-world costs of trying out policies in the target environment.

For these reasons, many researchers have investigated ways to automate the discovery of good representations (Dasgupta and McGregor, 1992; Radcliffe, 1993; Gruau, 1994; Stanley and Miikkulainen, 2002). Evolutionary methods are well suited to this challenge because they take a direct policy-search approach to reinforcement learning. In particular, since neuroevolution already directly searches the space of network weights, it can also simultaneously search the space of network topologies. Methods that do so are sometimes called *topology- and weight-evolving artificial neural networks* (TWEANNs).

Perhaps the earliest and simplest TWEANN is the *structured genetic algorithm* (sGA) (Dasgupta and McGregor, 1992), which uses a two-part representation to describe each network. The first part represents the connectivity of the network in the form of a binary matrix. Rows and columns correspond to nodes in the network and the value of each cell indicates whether an edge exists connecting the given pair of nodes. The second part represents the weights of each edge in the network. In principle, by evolving these binary matrices along with connection weights, sGA can automatically discover suitable network topologies. However, sGA suffers from several limitations. In the following section, we discuss these limitations in order to highlight the main challenges faced by all TWEANNs.

10.3.1 Challenges

There are three main challenges to developing a successful TWEANN. The first is the *competing conventions* problem. In most tasks, there are multiple different policies that have similar fitness. For example, many tasks contain symmetries that give rise to several equivalent solutions. This can lead to difficulties for evolution because of its reliance on crossover operators to breed new networks. When two networks that represent different policies are combined, the result is likely to be destructive, producing a policy that cannot successfully carry out the strategy used by either parent.

While competing conventions can arise in any evolutionary method that uses crossover, the problem is particularly severe for TWEANNs. Two parents may not only implement different policies but also have different representations. Therefore, to be effective, TWEANNs need a mechanism for combining networks with different topologies in a way that minimizes the chance of catastrophic crossover. Clearly, sGA does not meet this challenge, since the binary matrices it evolves are crossed over without regard to incompatibility in representations. In fact, the difficulties posed by the competing conventions problem were a major obstacle for early TWEANNs, to the point that some researchers simply avoided the problem by developing methods that do not perform crossover at all (Radcliffe, 1993).

The second challenge is the need to protect topological innovations long enough to optimize the associated weights. Typically, when new topological structures are introduced (e.g., the addition of a new hidden node or edge), it has a negative effect on fitness even if that structure will eventually be necessary for a good policy. The reason is that the weights associated with the new structure have not yet been optimized.

For example, consider an edge in a network evolved via sGA that is not activated, i.e., its cell in the binary matrix is set to zero. The corresponding weight for that edge will not experience any selective pressure, since it is not manifested in the network. If evolution suddenly activates that edge, the effect on fitness is likely to be detrimental, since its weight is not optimized. Therefore, if topological innovations are not explicitly protected, they will typically be eliminated from the population, causing the search for better topologies to stagnate.

Fortunately, protecting innovation is a well-studied problem in evolutionary computation. *Speciation* and *niching* methods (Holland, 1975; Goldberg and Richardson, 1987; Mahfoud, 1995; Potter and De Jong, 1995; Darwen and Yao, 1996) ensure diversity in the population, typically by segregating disparate individuals and/or penalizing individuals that are too similar to others. However, using such methods requires a distance metric to quantify the differences between individuals. Devising such a metric is difficult for TWEANNs, since it is not clear how to compare networks with different topologies.

The third challenge is how to evolve minimal solutions. As mentioned above, a central motivation for TWEANNs is the desire to avoid optimizing overly complex topologies. However, if evolution is initialized with a population of randomly chosen topologies, as in many TWEANNs, some of these topologies may already

be too complex. Thus, at least part of the evolutionary search will be conducted in an unnecessarily high dimensional space. It is possible to explicitly reward smaller solutions by adding size penalties in the fitness function (Zhang and Muhlenbein, 1993). However, there is no principled way to determine the size of the penalties without prior knowledge about the topological complexity required for the task.

10.3.2 NEAT

Perhaps the most popular TWEANN is *neuroevolution of augmenting topologies* (NEAT) (Stanley and Miikkulainen, 2002). In this section, we briefly describe NEAT and illustrate how it addresses the major challenges mentioned above.

NEAT is often used to evolve action selectors, as described in Section 10.2. In fact, NEAT follows the framework described in Algorithm 21 and differs from traditional neuroevolution only in how INIT-POPULATION and BREED-NET are implemented.

To represent networks of varying topologies, NEAT employs a flexible genetic encoding. Each network is described by a list of *edge genes*, each of which describes an edge between two *node genes*. Each edge gene specifies the in-node, the out-node, and the weight of the edge. During mutation, new structure can be introduced to a network via special mutation operators that add new node or edge genes to the network (see Figure 10.2).

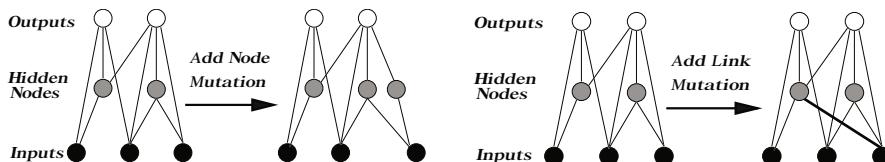


Fig. 10.2 Structural mutation operators in NEAT. At left, a new node is added by splitting an existing edge in two. At right, a new link (edge) is added between two existing nodes.

To avoid catastrophic crossover, NEAT relies on *innovation numbers*, which track the historical origin of each gene. Whenever a new gene appears via mutation, it receives a unique innovation number. Thus, the innovation numbers can be viewed as a chronology of all the genes produced during evolution.

During crossover, innovation numbers are used to determine which genes in the two parents correspond to each other. Genes that do not match are either *disjoint* or *excess*, depending on whether they occur within or outside the range of the other parent's innovation numbers. When crossing over, pairs of genes with the same innovation number (one from each parent) are lined up. Genes that do not match are inherited from the fitter parent. This approach makes it possible for NEAT to minimize the chance of catastrophic crossover without conducting an expensive

topological analysis. Since genomes with different topologies nonetheless remain compatible throughout evolution, NEAT essentially avoids the competing conventions problem.

Innovation numbers also make possible a simple way to protect topological innovation. In particular, NEAT uses innovation numbers to speciate the population based on topological similarity. The distance δ between two network encodings is a simple linear combination of the number of excess (E) and disjoint (D) genes, as well as the average weight differences of matching genes (\bar{W}):

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \bar{W}$$

The coefficients c_1 , c_2 , and c_3 adjust the importance of the three factors, and the factor N , the number of genes in the larger genome, normalizes for genome size. Networks whose distance is greater than δ_t , a compatibility threshold, are placed into different species. *Explicit fitness sharing* (Goldberg, 1989), in which networks in the same species must share the fitness of their niche, is employed to protect innovative species.

To encourage the evolution of minimal solutions, NEAT begins with a uniform population of simple networks with no hidden nodes and inputs connected directly to outputs. New structure is introduced incrementally via the mutation operators that add new hidden nodes and edges to the network. Since only the structural mutations that yield performance advantages tend to survive evolution's selective pressure, minimal solutions are favored.

NEAT has amassed numerous empirical successes on difficult reinforcement-learning tasks like non-Markovian double pole balancing (Stanley and Miikkulainen, 2002), game playing (Stanley and Miikkulainen, 2004b), and robot control (Stanley and Miikkulainen, 2004a; Taylor et al, 2006; Whiteson et al, 2010b). However, Kohl and Miikkulainen (2008, 2009) have shown that NEAT can perform poorly on tasks in which the optimal action varies discontinuously across states. They demonstrate that these problems can be mitigated by providing neurons with local receptive fields and constraining topology search to cascade structures

10.4 Hybrids

Many researchers have investigated hybrid methods that combine evolution with supervised or unsupervised learning methods. In such systems, the individuals being evolved do not remain fixed during their fitness evaluations. Instead, they change during their ‘lifetimes’ by learning from the environments with which they interact.

Much of the research on hybrid methods focuses on analyzing the dynamics that result when evolution and learning interact. For example, several studies (Whitley et al, 1994; Yamasaki and Sekiguchi, 2000; Pereira and Costa, 2001; Whiteson and Stone, 2006a) have used hybrids to compare *Lamarckian* and *Darwinian* systems.

In Lamarckian systems, the phenotypic effects of learning are copied back into the genome before reproduction, allowing new offspring to inherit them. In Darwinian systems, which more closely model biology, learning does not affect the genome. As other hybrid studies (Hinton and Nowlan, 1987; French and Messinger, 1994; Arita and Suzuki, 2000) have shown, Darwinian systems can indirectly transfer the results of learning into the genome by way of the *Baldwin effect* (Baldwin, 1896), in which learning creates selective pressures favoring individuals who innately possess attributes that were previously learned.

Hybrid methods have also been employed to improve performance on supervised learning tasks (Gruau and Whitley, 1993; Boers et al, 1995; Giraud-Carrier, 2000; Schmidhuber et al, 2005, 2007). However, such methods are not directly applicable to reinforcement-learning problems because the labeled data they require is absent.

Nonetheless, many hybrid methods for reinforcement learning have been developed. To get around the problem of missing labels, researchers have employed unsupervised learning (Stanley et al, 2003), trained individuals to resemble their parents (McQuesten and Miikkulainen, 1997), trained them to predict state transitions (Nolfi et al, 1994), and trained them to teach themselves (Nolfi and Parisi, 1997). However, perhaps the most natural hybrids for the reinforcement learning setting are combination of evolution with temporal-difference methods (Ackley and Littman, 1991; Wilson, 1995; Downing, 2001; Whiteson and Stone, 2006a). In this section, we survey two such hybrids: *evolutionary function approximation* and XCS, a type of *learning classifier system*.

10.4.1 Evolutionary Function Approximation

Evolutionary function approximation (Whiteson and Stone, 2006a), is a way to synthesize evolutionary and temporal-difference methods into a single method that automatically selects function approximator representations that enable efficient individual learning. The main idea is that, if evolution is directed to evolve value functions instead of action selectors, then those value functions can be updated, using temporal-difference methods, during each fitness evaluation. In this way, the system can *evolve* function approximators that are better able to *learn* via temporal-difference methods. This biologically intuitive combination, which has been applied to many computational systems (Hinton and Nowlan, 1987; Ackley and Littman, 1991; Boers et al, 1995; French and Messinger, 1994; Gruau and Whitley, 1993; Nolfi et al, 1994), can yield effective reinforcement-learning algorithms. In this section, we briefly describe NEAT+Q, an evolutionary function approximation technique resulting from the combination of NEAT and Q-learning with neural-network function approximation.

To make NEAT optimize value functions instead of action selectors, all that is required is a reinterpretation of its output values. The structure of neural-network action selectors (one input for each state feature and one output for each action) is already identical to that of Q-learning function approximators. Therefore, if the

weights of the networks NEAT evolves are updated during their fitness evaluations using Q-learning and backpropagation, they will effectively evolve value functions instead of action selectors. Hence, the outputs are no longer arbitrary values; they represent the long-term discounted values of the associated state-action pairs and are used, not just to select the most desirable action, but to update the estimates of other state-action pairs.

Algorithm 22 shows the inner loop of NEAT+Q, replacing lines 9–13 in Algorithm 21. Each time the agent takes an action, the network is backpropagated towards Q-learning targets (line 7) and ε -greedy selection occurs (lines 4–5). Figure 10.3 illustrates the complete algorithm: networks are selected from the population for evaluation and the Q-values they produce are used to select actions. The resulting feedback from the environment is used both to perform TD updates and to measure the network’s fitness, i.e., the total reward it accrues while learning.

```

1: //  $\alpha$ : learning rate,  $\gamma$ : discount factor,  $\lambda$ : eligibility decay rate,  $\varepsilon$ : exploration rate
2:
3:  $Q \leftarrow \text{EVAL-NET}(N, s')$                                 // compute value estimates for current state
4: with-prob( $\varepsilon$ )  $a' \leftarrow \text{RANDOM}(A)$                   // select random exploratory action
5: else  $a' \leftarrow \text{argmax}_k Q[k]$                          // or select greedy action
6: if  $s \neq \text{null}$  then
7:    $\text{BACKPROP}(N, s, a, (r + \gamma \max_k Q[k]), \alpha, \gamma, \lambda)$     // adjust weights
8:    $s, a \leftarrow s', a'$ 
9:    $r, s' \leftarrow \text{TAKE-ACTION}(a')$                            // take action and transition to new state
10:   $N.\text{fitness} \leftarrow N.\text{fitness} + r$                       // update total reward accrued by  $N$ 
```

Algorithm 22. NEAT+Q (inner loop)

Like other hybrid methods, NEAT+Q combines the advantages of temporal-difference methods with those of evolution. In particular, it harnesses the ability of NEAT to discover effective representations and uses it to aid neural-network value-function approximation. Unlike traditional neural-network function approximators, which put all their eggs in one basket by relying on a single manually designed network to represent the value function, NEAT+Q explores the space of such networks to increase the chance of finding a high performing representation. As a result, on certain tasks, this approach has been shown to significantly outperform both temporal-difference methods and neuroevolution on their own (Whiteson and Stone, 2006a).

10.4.2 XCS

A different type of hybrid method can be constructed using *learning classifier systems* (LCSs) (Holland, 1975; Holland and Reitman, 1977; Bull and Kovacs, 2005; Butz, 2006; Drugowitsch, 2008). An LCS is an evolutionary system that uses rules,

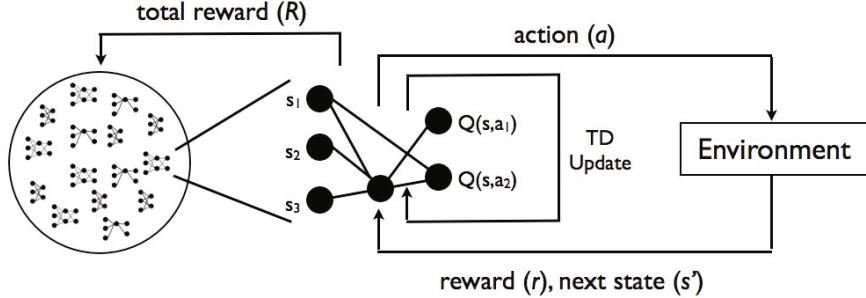


Fig. 10.3 The NEAT+Q algorithm

called *classifiers*, to represent solutions. A wide range of classification, regression, and optimization problems can be tackled using such systems. These include reinforcement learning problems, in which case a set of classifiers describes a policy.

Each classifier contains a *condition* that describes the set of states to which the classifier applies. Conditions can be specified in many ways (e.g., fuzzy conditions (Bonarini, 2000) or neural network conditions (Bull and O’Hara, 2002)) but perhaps the simplest is the ternary alphabet used for binary state features: 0, 1, and #, where # indicates “don’t care”. For example, a classifier with condition 01#10 applies to the states 01010 and 01110. Classifiers also contain *actions*, which specify what action the agent should take in states to which the classifier applies, and *predictions* which estimate the corresponding action-value function.

In *Pittsburgh-style* classifier systems (De Jong et al, 1993), each individual in the evolving population consists of an entire rule set describing a complete policy. These methods can be viewed as standard evolutionary approaches to reinforcement learning with rule sets as a way to represent policies, i.e., in lieu of neural networks. In contrast, in *Michigan-style* classifier systems, each individual consists of only one rule. Thus, the entire population represents a single, evolving policy. In the remainder of this section, we give a brief overview of XCS (Wilson, 1995, 2001; Butz et al, 2008), one of the most popular Michigan-style classifiers. Since prediction updates in XCS are based on Q-learning, it can be viewed as a hybrid between temporal-difference methods and evolutionary reinforcement-learning algorithms.

In XCS, each classifier’s prediction contributes to an estimate of the Q-values of the state-action pairs to which the classifier applies. $Q(s,a)$ is the weighted average of the predictions of all the *matching* classifiers, i.e., those that apply to s and have action a . The fitness of each classifier determines its weight in the average (fitness is defined later in this section). In other words:

$$Q(s,a) = \frac{\sum_{c \in M(s,a)} c.f \cdot c.p}{\sum_{c \in M(s,a)} c.f},$$

where $M(s,a)$ is the set of all classifiers matching s and a ; $c.f$ and $c.p$ are the fitness and prediction, respectively, of classifier c .

Each time the agent is in state s , takes action a , receives reward r , and transitions to state s' , the following update rule is applied to each $c \in M(s,a)$:

$$c.p \leftarrow c.p + \beta [r + \gamma \max_{a'} Q(s', a') - c.p] \frac{c.f}{\sum_{c' \in M(s,a)} c'.f},$$

where β is a learning rate parameter. This is essentially a Q-learning update, except that the size of the update is scaled according to the relative fitness of c , since this determines its contribution to the Q-value.

Many LCS systems use *strength-based* updates, wherein each classifier's fitness is based on its prediction, i.e., classifiers that expect to obtain more reward are favored by evolution. However, this can lead to the problem of *strong over-generals* (Kovacs, 2003), which are general classifiers that have high overall value but select suboptimal actions for a subset of states. With strength-based updates, such classifiers are favored over more specific ones that select better actions for that same subset but have lower overall value.

To avoid this problem, XCS uses *accuracy-based* updates, in which a classifier's fitness is inversely proportional to an estimate of the error in its prediction. This error $c.\epsilon$ is updated based on the absolute value of the temporal-difference error used in the Q-learning update:

$$c.\epsilon \leftarrow c.\epsilon + \beta (|r + \gamma \max_{a'} Q(s', a') - c.p| - c.\epsilon)$$

Classifier accuracy is then defined in terms of this error. Specifically, when $c.\epsilon > \epsilon_0$, a minimum error threshold, the accuracy of c is defined as:

$$c.\kappa = \alpha (c.\epsilon / \epsilon_0)^{-\eta},$$

where α and η are accuracy parameters. When $c.\epsilon \leq \epsilon_0$, $c.\kappa = 1$.

However, fitness is computed, not with this accuracy, but instead with the *set-relative accuracy*: the accuracy divided by the sum of the accuracies of all matching classifiers. This yields the following fitness update:

$$c.f \leftarrow c.f + \beta \left(\frac{c.\kappa}{\sum_{c' \in M(s,a)} c'.\kappa} - c.f \right)$$

At every timestep, the prediction, error and fitness of each matching classifier are updated. Since XCS is a *steady-state* evolutionary method, there are no generations. Instead, the population changes incrementally through the periodic selection and reproduction of a few fit classifiers, which replace a few weak classifiers and leave the rest of the population unchanged. When selection occurs, only classifiers that match the current state and action are considered. New classifiers are created from the selected ones using crossover and mutation, and weak classifiers (chosen from the whole population, not just the matching ones) are deleted to make room.

Thanks to the Q-learning updates, the accuracy of the classifiers tends to improve over time. Thanks to steady-state evolution, the most accurate classifiers are selectively bred. General rules tend to have higher error (since they generalize over more

states) and thus lower accuracy. It might seem that, as a result, XCS will evolve only highly specific rules. However, more general rules also match more often. Since only matching classifiers can reproduce, XCS balances the pressure for specific rules with pressure for general rules. Thus, it strives to learn a complete, maximally general, and accurate set of classifiers for approximating the optimal Q-function.

Though there are no convergence proofs for XCS on MDPs, it has proven empirically effective on many tasks. For example, on maze tasks, it has proven adept at automatically discovering what state features to ignore (Butz et al, 2005) and solving problems with more than a million states (Butz and Lanzi, 2009). It has also proven adept at complex sensorimotor control (Butz and Herbort, 2008; Butz et al, 2009) and autonomous robotics (Dorigo and Colombetti, 1998).

10.5 Coevolution

Coevolution is a concept from evolutionary biology that refers to the interactions between multiple individuals that are simultaneously evolving. In other words, coevolution occurs when the fitness function of one individual depends on other individuals that are also evolving. In nature, this can occur when different populations interact, e.g., cheetahs and the gazelles they hunt, or within a population, e.g., members of the same species competing for mates. Furthermore, coevolution can be cooperative, e.g., humans and the bacteria in our digestive systems, or competitive, e.g., predator and prey. All these forms of coevolution have been investigated and exploited in evolutionary reinforcement learning, as surveyed in this section.

10.5.1 Cooperative Coevolution

The most obvious application of cooperative coevolution is to cooperative multi-agent systems (Panait and Luke, 2005) which, in the context of reinforcement learning, means evolving teams of agents that coordinate their behavior to solve a sequential decision problem. In principle, such problems can be solved without coevolution by using a monolithic approach: evolving a population in which each individual specifies the policy for every agent on the team. However, such an approach quickly becomes intractable, as the size of the search space grows exponentially with respect to the number of agents.

One of the primary motivations for a coevolutionary approach is that it can help address this difficulty (Wiegand et al, 2001; Jansen and Wiegand, 2004; Panait et al, 2006). As Gomez et al. put it, “many problems may be decomposable into weakly coupled low-dimensional subspaces that can be searched semi-independently by separate species” (Gomez et al, 2008). In general, identifying these low-dimensional subspaces requires a lot of domain knowledge. However, in multi-agent problems, it is often sufficient to divide the problem up by agent, i.e., evolve one population for each agent on the team, in order to make evolution tractable. In this approach,

one member of each population is selected, often randomly, to form a team that is then evaluated in the task. The total reward obtained contributes to an estimate of the fitness of each participating agent, which is typically evaluated multiple times.

While this approach often outperforms monolithic evolution and has found success in predator-prey (Yong and Miikkulainen, 2007) and robot-control (Cai and Peng, 2002) applications, it also runs into difficulties when there are large numbers of agents. The main problem is that the contribution of a single agent to the total reward accrued becomes insignificant. Thus, the fitness an agent receives depends more on which teammates it is evaluated with than on its own policy. However, it is possible to construct special fitness functions for individual agents that are much less sensitive to such effects (Agogino and Tumer, 2008). The main idea is to use *difference functions* (Wolpert and Tumer, 2002) that compare the total reward the team obtains when the agent is present to when it is absent or replaced by a fixed baseline agent. While this approach requires access to a model of the environment and increases the computational cost of fitness evaluation (so that the reward in both scenarios can be measured), it can dramatically improve the performance of cooperative coevolution.

Coevolution can also be used to simultaneously evolve multiple components of a single agent, instead of multiple agents. For example, in the task of robot soccer keepaway, domain knowledge has been used to decompose the task into different components, each representing an important skill such as running towards the ball or getting open for a pass (Whiteson et al, 2005). Neural networks for each of these components are then coevolved and together comprise a complete policy. In the keepaway task, coevolution greatly outperforms a monolithic approach.

Cooperative coevolution can also be used in a single-agent setting to facilitate neuroevolution. Rather than coevolving multiple networks, with one for each member of a team or each component of a policy, *neurons* are coevolved, which together form a single network describing the agent's policy (Potter and De Jong, 1995, 2000). Typically, networks have fixed topologies with a single hidden layer and each neuron corresponds to a hidden node, including all the weights of its incoming and outgoing edges. Just as dividing a multi-agent task up by agent often leads to simpler subproblems, so too can breaking up a neuroevolutionary task by neuron. As Moriarty and Miikkulainen say, “neuron-level evolution takes advantage of the *a priori* knowledge that individual neurons constitute basic components of neural networks” (Moriarty and Miikkulainen, 1997).

One example is *symbiotic adaptive neuroevolution* (SANE) (Moriarty and Miikkulainen, 1996, 1997) in which evolution occurs simultaneously at two levels. At the lower level, a single population of neurons is evolved. The fitness of each neuron is the average performance of the networks in which it participates during fitness evaluations. At the higher level, a population of *blueprints* is evolved, with each blueprint consisting of a vector of pointers to neurons in the lower level. The blueprints that combine neurons into the most effective networks tend to survive selective pressure. On various reinforcement-learning tasks such as robot control and

pole balancing, SANE has outperformed temporal-difference methods, monolithic neuroevolution, and neuron-level evolution without blueprints.

The *enforced subpopulations* (ESP) method (Gomez and Miikkulainen, 1999) eliminates the blueprint population but segregates neurons into subpopulations, one for each hidden node. One member of each population is selected randomly to form a network for each fitness evaluation. This encourages subpopulations to take on different specialized roles, increasing the likelihood that effective networks will be formed even without blueprints. ESP has performed particularly well on partially observable tasks, solving a non-Markovian version of the double pole-balancing problem. In addition H-ESP, a hierarchical variant that does use blueprints, has proven successful on *deep memory POMDPs*, i.e., those requiring history of hundreds or thousands of timesteps (Gomez and Schmidhuber, 2005a). ESP has even been used to evolve control systems for a finless sounding rocket (Gomez and Miikkulainen, 2003).

In *cooperative synapse neuroevolution* (CoSyNE) (Gomez et al, 2006, 2008), the idea of separate subpopulations is taken even further. Rather than a subpopulation for every neuron, which contains multiple weights, CoSyNE has a subpopulation for each edge, which has only one weight (see Figure 10.4). Thus the problem of finding a complete network is broken down into atomic units, solutions to which are coevolved. On several versions of the pole balancing problem, CoSyNE has been shown to outperform various temporal-difference methods and other policy-search approaches, as well as SANE, ESP, and NEAT (Gomez et al, 2006, 2008). However, in a more recent study, CMA-NeuroES (see Section 10.2) performed even better (Heidrich-Meisner and Igel, 2009b).

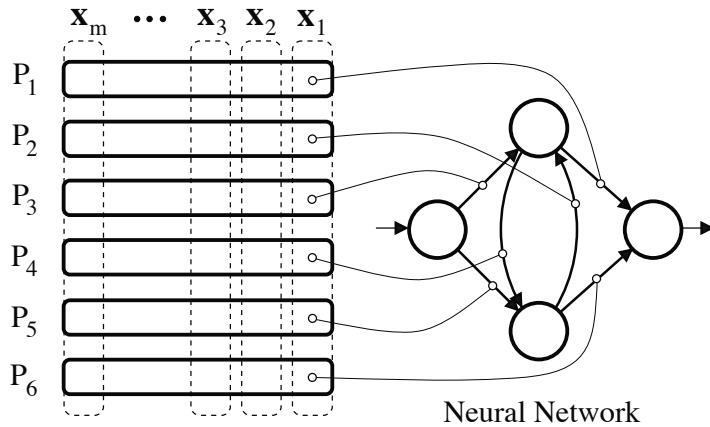


Fig. 10.4 The CoSyNE algorithm, using six subpopulations, each containing m weights. All the weights at a given index i form a genotype \mathbf{x}_i . Each weight is taken from a different subpopulation and describes one edge in the neural network. Figure taken with permission from (Gomez et al, 2008).

10.5.2 Competitive Coevolution

Coevolution has also proven a powerful tool for competitive settings. The most common applications are in games, in which coevolution is used to simultaneously evolve strong players and the opponents against which they are evaluated. The hope is to create an *arms race* (Dawkins and Krebs, 1979) in which the evolving agents exert continual selective pressure on each other, driving evolution towards increasingly effective policies.

Perhaps the simplest example of competitive coevolution is the work of Pollack and Blair in the game of backgammon (Pollack and Blair, 1998). Their approach relies on a simple optimization technique (essentially an evolutionary method with a population size of two) wherein a neural network plays against a mutated version of itself and the winner survives. The approach works so well that Pollack and Blair hypothesize that Tesauro's great success with TD-Gammon (Tesauro, 1994) is due more to the nature of backgammon than the power of temporal-difference methods.²

Using larger populations, competitive coevolution has also found success in the game of checkers. The Blondie24 program uses the *minimax* algorithm (Von Neumann, 1928) to play checkers, relying on neuroevolution to discover an effective evaluator of board positions (Chellapilla and Fogel, 2001). During fitness evaluations, members of the current population play games against each other. Despite the minimal use of human expertise, Blondie24 evolved to play at a level competitive with human experts.

Competitive coevolution can also have useful synergies with TWEANNs. In fixed-topology neuroevolution, arms races may be cut short when additional improvement requires an expanded representation. Since TWEANNs can automatically expand their representations, coevolution can give rise to *continual complication* (Stanley and Miikkulainen, 2004a).

The methods mentioned above evolve only a single population. However, as in cooperative coevolution, better performance is sometimes possible by segregating individuals into separate populations. In the *host/parasite* model (Hillis, 1990), one population evolves *hosts* and another *parasites*. Hosts are evaluated based on their robustness against parasites, e.g., how many parasites they beat in games of checkers. In contrast, parasites are evaluated based on their uniqueness, e.g., how many hosts they can beat that other parasites cannot. Such fitness functions can be implemented using *competitive fitness sharing* (Rosin and Belew, 1997).

In *Pareto coevolution*, the problem is treated as a multi-objective one, with each opponent as an objective (Ficici and Pollack, 2000, 2001). The goal is thus to find a *Pareto-optimal solution*, i.e., one that cannot be improved with respect to one objective without worsening its performance with respect to another. Using this approach, many methods have been developed that maintain *Pareto archives* of

² Tesauro, however, disputes this claim, pointing out that the performance difference between Pollack and Blair's approach and his own is quite significant, analogous to that between an average human player and a world-class one (Tesauro, 1998).

opponents against which to evaluate evolving solutions (De Jong, 2004; Monroy et al, 2006; De Jong, 2007; Popovici et al, 2010).

10.6 Generative and Developmental Systems

All of the evolutionary reinforcement-learning methods described above rely on *direct encodings* to represent policies. In such representations, evolution optimizes a *genotype* (e.g., a vector of numbers specifying the weights of a neural network) that can be trivially transformed into a *phenotype* (e.g., the neural network itself). While the simplicity of such an approach is appealing, it has limited scalability. Since the genotype is always as large as the phenotype, evolving the complex policies required to solve many realistic tasks requires searching a high dimensional space.

Generative and developmental systems (Gruau, 1994; Hornby and Pollack, 2002; Stanley and Miikkulainen, 2003; Stanley et al, 2009) is a subfield of evolutionary computation that focuses on evolving *indirect encodings*. In such representations, the phenotype is ‘grown’ via a developmental process specified by the genotype. In many cases, good policies possess simplifying regularities such as symmetry and repetition, which allow for genotypes that are much smaller than the phenotypes they produce. Consequently, searching genotype space is more feasible, making it possible to scale to larger reinforcement-learning tasks.

Furthermore, indirect encodings often provide a natural way to exploit a task’s *geometry*, i.e., the spatial relationship between state features. In most direct encodings, such geometry cannot be exploited because it is not captured in the representation. For example, consider a neural network in which each input describes the current state of one square on a chess board. Since these inputs are treated as an unordered set, the distance between squares is not captured in the representation. Thus, structures for exploiting the relationship between squares must be evolved, complicating the task. In contrast, an indirect encoding can describe a network where the structure for processing each square’s state is a function of that square’s position on the board, with the natural consequence that nearby squares are processed similarly.

Like other evolutionary methods, systems using indirect encodings are inspired by analogies with biological systems: e.g., human beings have trillions of cells in their bodies but develop from genomes containing only tens of thousands of genes. Thus, it is not surprising that many indirect encodings are built on models of natural development. For example, *L-systems* (Lindenmayer, 1968), which are formal grammars for describing complex recursive structures, have been used to evolve both the morphology and control system for autonomous robots, greatly outperforming direct encodings (Hornby and Pollack, 2002). Similarly, *cellular encodings* (Gruau and Whitley, 1993; Gruau, 1994) evolve graph grammars for generating modular neural networks composed of simpler subnetworks. This approach allows evolution to exploit regularities in the solution structure by instantiating multiple copies of the same subnetwork in order to build a complete network.

More recently, the HyperNEAT method (Stanley et al, 2009) has been developed to extend NEAT to use indirect encodings. This approach is based on *compositional pattern producing networks* (CPPNs). CPPNs are neural networks for describing complex patterns. For example, a two-dimensional image can be described by a CPPN whose inputs correspond to an x - y position in the image and whose output corresponds to the color that should appear in that position. The image can then be generated by querying the CPPN at each x - y position and setting that position's color based on the output. Such CPPNs can be evolved by NEAT, yielding a developmental system with the CPPN as the genotype and the image as the phenotype.

In HyperNEAT, the CPPN is used to describe a neural network instead of an image. Thus, both the genotype and phenotype are neural networks. As illustrated in Figure 10.5, the nodes of the phenotypic network are laid out on a *substrate*, i.e., a grid, such that each has a position. The CPPN takes as input two positions instead of one and its output specifies the weight of the edge connecting the two corresponding nodes. As before, these CPPNs can be evolved by NEAT based on the fitness of the resulting phenotypic network, e.g., its performance as a policy in a reinforcement learning task. The CPPNs can be interpreted as describing a spatial pattern in a four-dimensional hypercube, yielding the name HyperNEAT. Because the developmental approach makes it easy to specify networks that exploit symmetries and regularities in complex tasks, HyperNEAT has proven an effective tool for reinforcement learning, with successful applications in domains such as checkers (Gauci and Stanley, 2008, 2010), keepaway soccer (Verbancsics and Stanley, 2010), and multi-agent systems (D'Ambrosio et al, 2010).

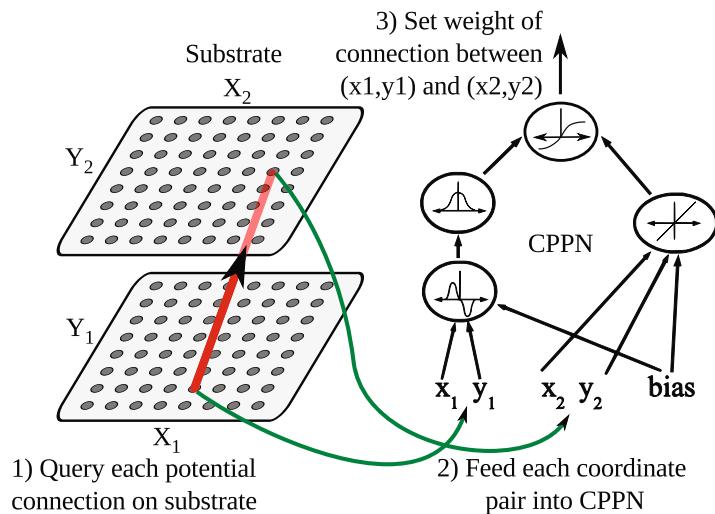


Fig. 10.5 The HyperNEAT algorithm. Figure taken with permission from (Gauci and Stanley, 2010).

10.7 On-Line Methods

While evolutionary methods have excelled in many challenging reinforcement-learning problems, their empirical success is largely restricted to *off-line* scenarios, in which the agent learns, not in the real-world, but in a ‘safe’ environment like a simulator. In other words, the problem specification usually includes a fitness function that requires only computational resources to evaluate, as in other optimization problems tackled with evolutionary computation.

In off-line scenarios, an agent’s only goal is to learn a good policy as quickly as possible. How much reward it obtains *while it is learning* is irrelevant because those rewards are only hypothetical and do not correspond to real-world costs. If the agent tries disastrous policies, only computation time is lost.

While efficient off-line learning is an important goal, it has limited practical applicability. In many cases, no simulator is available because the dynamics of the task are unknown, e.g., when a robot explores an unfamiliar environment or a chess player plays a new opponent. Other times, the dynamics of the task are too complex to accurately simulate, e.g., user behavior on a large computer network or the noise in a robot’s sensors and actuators.

Therefore, many researchers consider *on-line* learning a fundamental challenge in reinforcement learning. In an on-line learning scenario, the agent must maximize the reward it accrues while it is learning because those rewards correspond to real-world costs. For example, if a robot learning on-line tries a policy that causes it to drive off a cliff, then the negative reward the agent receives is not hypothetical; it corresponds to the real cost of fixing or replacing the robot.

Evolutionary methods have also succeeded on-line (Steels, 1994; Nordin and Banzhaf, 1997; Schroder et al, 2001), especially in evolutionary robotics (Meyer et al, 1998; Floreano and Urzelai, 2001; Floreano and Mondada, 2002; Pratihar, 2003; Kernbach et al, 2009; Zufferey et al, 2010), and some research has investigated customizing such methods to on-line settings (Floreano and Urzelai, 2001; Whiteson and Stone, 2006a,b; Priesterjahn et al, 2008; Tan et al, 2008; Cardamone et al, 2009, 2010).

Nonetheless, in most applications, researchers typically report performance using only the off-line measures common for optimization problems, e.g., the number of fitness evaluations needed to find a policy achieving a threshold performance or the performance of the best policy found after a given number of fitness evaluations. Therefore, determining how best to use evolutionary methods for reinforcement learning in on-line settings, i.e., how to maximize cumulative reward during evolution, remains an important and under-explored research area.

10.7.1 Model-Based Methods

One possible approach is to use evolution, not as a complete solution method, but as a component in a model-based method. In model-based algorithms, the agent’s

interactions with its environment are used to learn a model, to which planning methods are then applied. As the agent gathers more samples from the environment, the quality of the model improves, which, in turn, improves the quality of the policy produced via planning. Because planning is done off-line, the number of interactions needed to find a good policy is minimized, leading to strong on-line performance.

In such an approach, planning is typically conducted using dynamic programming methods like value iteration. However, many other methods can be used instead; if the model is continuous and/or high dimensional, evolutionary or other policy-search methods may be preferable. Unfortunately, most model-based methods are designed only to learn tabular models for small, discrete state spaces. Still, in some cases, especially when considerable domain expertise is available, more complex models can be learned.

For example, linear regression has been used to learn models of helicopter dynamics, which can then be used for policy-search reinforcement learning (Ng et al, 2004). The resulting policies have successfully controlled real model helicopters. A similar approach was used to maximize on-line performance in the helicopter-hovering events in recent Reinforcement Learning Competitions (Whiteson et al, 2010a); models learned via linear regression were used as fitness functions for policies evolved off-line via neuroevolution (Koppejan and Whiteson, 2009).

Alternatively, evolutionary methods can be used for the model-learning component of a model-based solution. In particular, *anticipatory learning classifier systems* (Butz, 2002; Gerard et al, 2002, 2005; Sigaud et al, 2009), a type of LCS, can be used to evolve models of the environment that are used for planning in a framework similar to Dyna-Q (Sutton, 1990).

10.7.2 On-Line Evolutionary Computation

Another possible solution is *on-line evolutionary computation* (Whiteson and Stone, 2006a,b). The main idea is to borrow exploration strategies commonly used to select actions in temporal-difference methods and use them to select policies for evaluation in evolution. Doing so allows evolution to balance exploration and exploitation in a way that improves on-line performance.

Of course, evolutionary methods already strive to balance exploration and exploitation. In fact, this is one of the main motivations originally provided for genetic algorithms (Holland, 1975). However, this balance typically occurs only *across* generations, not *within* them. Once the members of each generation have been determined, they all typically receive the same evaluation time.

This approach makes sense in deterministic domains, where each member of the population can be accurately evaluated in a single episode. However, many real-world domains are stochastic, in which case fitness evaluations must be averaged over many episodes. In these domains, giving the same evaluation time to each member of the population can be grossly suboptimal because, within a generation, it is purely exploratory.

Instead, on-line evolutionary computation exploits information gained earlier in the generation to systematically give more evaluations to more promising policies and avoid re-evaluating weaker ones. This is achieved by employing temporal-difference exploration strategies to select policies for evaluation in each generation.

For example, ϵ -greedy selection can be used at the beginning of each episode to select a policy for evaluation. Instead of iterating through the population, evolution selects a policy randomly with probability ϵ . With probability $1 - \epsilon$, the algorithm selects the best policy discovered so far in the current generation. The fitness of each policy is just the average reward per episode it has received so far. Each time a policy is selected for evaluation, the total reward it receives is incorporated into that average, which can cause it to gain or lose the rank of best policy.

For the most part, ϵ -greedy selection does not alter evolution's search but simply interleaves it with exploitative episodes that increase average reward during learning. However, softmax selection can also be used to focus exploration on the most promising alternatives. At the beginning of each generation, each individual is evaluated for one episode, to initialize its fitness. Then, the remaining $e - |P|$ episodes are allocated according to a Boltzmann distribution.

Neither ϵ -greedy nor softmax consider the uncertainty of the estimates on which they base their selections, a shortcoming that can be addressed with interval estimation (Kaelbling, 1993). When used in temporal-difference methods, interval estimation computes a $(100 - \alpha)\%$ confidence interval for the value of each available action. The agent always takes the action with the highest upper bound on this interval. This strategy favors actions with high estimated value and also focuses exploration on promising but uncertain actions. The α parameter controls the balance between exploration and exploitation, with smaller values generating greater exploration.

The same strategy can be employed within evolution to select policies for evaluation. At the beginning of each generation, each individual is evaluated for one episode, to initialize its fitness. Then, the remaining $e - |P|$ episodes are allocated to the policy that currently has the highest upper bound on its confidence interval.

All three of these implementations of on-line evolutionary computation have been shown to substantially improve on-line performance, both in conjunction with NEAT (Whiteson and Stone, 2006b; Cardamone et al, 2009, 2010) and NEAT+Q (Whiteson and Stone, 2006a).

10.8 Conclusion

Evolutionary methods are a powerful tool for tackling challenging reinforcement learning problems. They are especially appealing for problems that include partial observability, have continuous action spaces, or where effective representations cannot be manually specified. Particularly in the area of neuroevolution, sophisticated methods exist for evolving neural-network topologies, decomposing the task based on network structure, and exploiting indirect encodings. Thanks to hybrid

methods, the use of evolutionary computation does not require forgoing the power of temporal-difference methods. Furthermore, coevolutionary approaches extend the reach of evolution to multi-agent reinforcement learning, both cooperative and competitive. While most work in evolutionary computation has focused on off-line settings, promising research exists in developing evolutionary methods for on-line reinforcement learning, which remains a critical and exciting challenge for future work.

Acknowledgements. Thanks to Ken Stanley, Risto Miikkulainen, Jürgen Schmidhuber, Martin Butz, Julian Bishop, and the anonymous reviewers for their invaluable input regarding the state of the art in evolutionary reinforcement learning.

References

- Ackley, D., Littman, M.: Interactions between learning and evolution. *Artificial Life II, SFI Studies in the Sciences of Complexity* 10, 487–509 (1991)
- Agogino, A.K., Turner, K.: Efficient evaluation functions for evolving coordination. *Evolutionary Computation* 16(2), 257–288 (2008)
- Arita, T., Suzuki, R.: Interactions between learning and evolution: The outstanding strategy generated by the Baldwin Effect. *Artificial Life* 7, 196–205 (2000)
- Baldwin, J.M.: A new factor in evolution. *The American Naturalist* 30, 441–451 (1896)
- Boers, E., Borst, M., Sprinkhuizen-Kuyper, I.: Evolving Artificial Neural Networks using the “Baldwin Effect”. In: *Proceedings of the International Conference Artificial Neural Nets and Genetic Algorithms* in Ales, France (1995)
- Bonarini, A.: An introduction to learning fuzzy classifier systems. *Learning Classifier Systems*, 83–104 (2000)
- Bull, L., Kovacs, T.: Foundations of learning classifier systems: An introduction. *Foundations of Learning Classifier Systems*, 1–17 (2005)
- Bull, L., O’Hara, T.: Accuracy-based neuro and neuro-fuzzy classifier systems. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 905–911 (2002)
- Butz, M.: Anticipatory learning classifier systems. *Kluwer Academic Publishers* (2002)
- Butz, M.: Rule-based evolutionary online learning systems: A principled approach to LCS analysis and design. *Springer, Heidelberg* (2006)
- Butz, M., Herbort, O.: Context-dependent predictions and cognitive arm control with XCSF. In: *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pp. 1357–1364. ACM (2008)
- Butz, M., Lanzi, P.: Sequential problems that test generalization in learning classifier systems. *Evolutionary Intelligence* 2(3), 141–147 (2009)
- Butz, M., Goldberg, D., Lanzi, P.: Gradient descent methods in learning classifier systems: Improving XCS performance in multistep problems. *IEEE Transactions on Evolutionary Computation* 9(5) (2005)
- Butz, M., Lanzi, P., Wilson, S.: Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction. *IEEE Transactions on Evolutionary Computation* 12(3), 355–376 (2008)

- Butz, M., Pedersen, G., Stalp, P.: Learning sensorimotor control structures with XCSF: Redundancy exploitation and dynamic control. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 1171–1178 (2009)
- Cai, Z., Peng, Z.: Cooperative coevolutionary adaptive genetic algorithm in path planning of cooperative multi-mobile robot systems. *Journal of Intelligent and Robotic Systems* 33(1), 61–71 (2002)
- Cardamone, L., Loiacono, D., Lanzi, P.: On-line neuroevolution applied to the open racing car simulator. In: Proceedings of the Congress on Evolutionary Computation (CEC), pp. 2622–2629 (2009)
- Cardamone, L., Loiacono, D., Lanzi, P.L.: Learning to drive in the open racing car simulator using online neuroevolution. *IEEE Transactions on Computational Intelligence and AI in Games* 2(3), 176–190 (2010)
- Chellapilla, K., Fogel, D.: Evolving an expert checkers playing program without using human expertise. *IEEE Transactions on Evolutionary Computation* 5(4), 422–428 (2001)
- Coello, C., Lamont, G., Van Veldhuizen, D.: Evolutionary algorithms for solving multi-objective problems. Springer, Heidelberg (2007)
- D'Ambrosio, D., Lehman, J., Risi, S., Stanley, K.O.: Evolving policy geometry for scalable multiagent learning. In: Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), pp. 731–738 (2010)
- Darwen, P., Yao, X.: Automatic modularization by speciation. In: Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC 1996), pp. 88–93 (1996)
- Dasgupta, D., McGregor, D.: Designing application-specific neural networks using the structured genetic algorithm. In: Proceedings of the International Conference on Combinations of Genetic Algorithms and Neural Networks, pp. 87–96 (1992)
- Dawkins, R., Krebs, J.: Arms races between and within species. *Proceedings of the Royal Society of London Series B, Biological Sciences* 205(1161), 489–511 (1979)
- de Jong, E.D.: The Incremental Pareto-coevolution Archive. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 525–536. Springer, Heidelberg (2004)
- De Jong, E.: A monotonic archive for Pareto-coevolution. *Evolutionary Computation* 15(1), 61–93 (2007)
- De Jong, K., Spears, W.: An analysis of the interacting roles of population size and crossover in genetic algorithms. In: Parallel Problem Solving from Nature, pp. 38–47 (1991)
- De Jong, K., Spears, W., Gordon, D.: Using genetic algorithms for concept learning. *Machine learning* 13(2), 161–188 (1993)
- Deb, K.: Multi-objective optimization using evolutionary algorithms. Wiley (2001)
- Dorigo, M., Colombetti, M.: Robot shaping: An experiment in behavior engineering. The MIT Press (1998)
- Downing, K.L.: Reinforced genetic programming. *Genetic Programming and Evolvable Machines* 2(3), 259–288 (2001)
- Doya, K.: Reinforcement learning in continuous time and space. *Neural Computation* 12(1), 219–245 (2000)
- Drugowitsch, J.: Design and analysis of learning classifier systems: A probabilistic approach. Springer, Heidelberg (2008)
- Ficici, S., Pollack, J.: A game-theoretic approach to the simple coevolutionary algorithm. In: Parallel Problem Solving from Nature PPSN VI, pp. 467–476. Springer, Heidelberg (2000)
- Ficici, S., Pollack, J.: Pareto optimality in coevolutionary learning. *Advances in Artificial Life*, 316–325 (2001)

- Floreano, D., Mondada, F.: Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 26(3), 396–407 (2002)
- Floreano, D., Urzelai, J.: Evolution of plastic control networks. *Autonomous Robots* 11(3), 311–317 (2001)
- French, R., Messinger, A.: Genes, phenes and the Baldwin effect: Learning and evolution in a simulated population. *Artificial Life* 4, 277–282 (1994)
- Gaskett, C., Wettergreen, D., Zelinsky, A.: Q-learning in continuous state and action spaces. *Advanced Topics in Artificial Intelligence*, 417–428 (1999)
- Gauci, J., Stanley, K.O.: A case study on the critical role of geometric regularity in machine learning. In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, AAAI 2008 (2008)
- Gauci, J., Stanley, K.O.: Autonomous evolution of topographic regularities in artificial neural networks. *Neural Computation* 22(7), 1860–1898 (2010)
- Gerard, P., Stolzmann, W., Sigaud, O.: YACS: a new learning classifier system using anticipation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications* 6(3), 216–228 (2002)
- Gerard, P., Meyer, J., Sigaud, O.: Combining latent learning with dynamic programming in the modular anticipatory classifier system. *European Journal of Operational Research* 160(3), 614–637 (2005)
- Giraud-Carrier, C.: Unifying learning with evolution through Baldwinian evolution and Lamarckism: A case study. In: *Proceedings of the Symposium on Computational Intelligence and Learning* (CoIL 2000), pp. 36–41 (2000)
- Goldberg, D.: *Genetic Algorithms in Search*. In: *Optimization and Machine Learning*, Addison-Wesley (1989)
- Goldberg, D., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms* 1, 69–93 (1991)
- Goldberg, D., Richardson, J.: Genetic algorithms with sharing for multimodal function optimization. In: *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, p. 49 (1987)
- Gomez, F., Miikkulainen, R.: Solving non-Markovian control tasks with neuroevolution. In: *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1356–1361 (1999)
- Gomez, F., Miikkulainen, R.: Active guidance for a finless rocket using neuroevolution. In: *GECCO 2003: Proceedings of the Genetic and Evolutionary Computation Conference* (2003)
- Gomez, F., Schmidhuber, J.: Co-evolving recurrent neurons learn deep memory POMDPs. In: *GECCO 2005: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 491–498 (2005a)
- Gomez, F.J., Schmidhuber, J.: Evolving Modular Fast-Weight Networks for Control. In: Duch, W., Kacprzyk, J., Oja, E., Zadrożny, S. (eds.) *ICANN 2005*. LNCS, vol. 3697, pp. 383–389. Springer, Heidelberg (2005b)
- Gomez, F.J., Schmidhuber, J., Miikkulainen, R.: Efficient Non-Linear Control Through Neuroevolution. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) *ECML 2006*. LNCS (LNAI), vol. 4212, pp. 654–662. Springer, Heidelberg (2006)
- Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research* 9, 937–965 (2008)
- Gruau, F.: Automatic definition of modular neural networks. *Adaptive Behavior* 3(2), 151 (1994)

- Gruau, F., Whitley, D.: Adding learning to the cellular development of neural networks: Evolution and the Baldwin effect. *Evolutionary Computation* 1, 213–233 (1993)
- Hansen, N., Müller, S., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation* 11(1), 1–18 (2003)
- van Hasselt, H., Wiering, M.: Reinforcement learning in continuous action spaces. In: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, ADPRL, pp. 272–279 (2007)
- Haykin, S.: Neural networks: a comprehensive foundation. Prentice-Hall (1994)
- Heidrich-Meisner, V., Igel, C.: Variable metric reinforcement learning methods applied to the noisy mountain car problem. *Recent Advances in Reinforcement Learning*, 136–150 (2008)
- Heidrich-Meisner, V., Igel, C.: Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 401–408 (2009a)
- Heidrich-Meisner, V., Igel, C.: Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms* 64(4), 152–168 (2009b)
- Heidrich-Meisner, V., Igel, C.: Uncertainty handling CMA-ES for reinforcement learning. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 1211–1218 (2009c)
- Hillis, W.: Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena* 42(1-3), 228–234 (1990)
- Hinton, G.E., Nowlan, S.J.: How learning can guide evolution. *Complex Systems* 1, 495–502 (1987)
- Holland, J., Reitman, J.: Cognitive systems based on adaptive algorithms. *ACM SIGART Bulletin* 63, 49–49 (1977)
- Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology. In: Control and Artificial Intelligence. University of Michigan Press (1975)
- Hornby, G., Pollack, J.: Creating high-level components with a generative representation for body-brain evolution. *Artificial Life* 8(3), 223–246 (2002)
- Igel, C.: Neuroevolution for reinforcement learning using evolution strategies. In: Congress on Evolutionary Computation, vol. 4, pp. 2588–2595 (2003)
- Jansen, T., Wiegand, R.P.: The cooperative coevolutionary (1+1) EA. *Evolutionary Computation* 12(4), 405–434 (2004)
- Kaelbling, L.P.: Learning in Embedded Systems. MIT Press (1993)
- Kernbach, S., Meister, E., Scholz, O., Humza, R., Liedke, J., Ricotti, L., Jemai, J., Havlik, J., Liu, W.: Evolutionary robotics: The next-generation-platform for on-line and on-board artificial evolution. In: CEC 2009: IEEE Congress on Evolutionary Computation, pp. 1079–1086 (2009)
- Kohl, N., Miikkulainen, R.: Evolving neural networks for fractured domains. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1405–1412 (2008)
- Kohl, N., Miikkulainen, R.: Evolving neural networks for strategic decision-making problems. *Neural Networks* 22, 326–337 (2009); (special issue on Goal-Directed Neural Systems)
- Koppejan, R., Whiteson, S.: Neuroevolutionary reinforcement learning for generalized helicopter control. In: GECCO 2009: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 145–152 (2009)

- Kovacs, T.: Strength or accuracy: credit assignment in learning classifier systems. Springer, Heidelberg (2003)
- Larranaga, P., Lozano, J.: Estimation of distribution algorithms: A new tool for evolutionary computation. Springer, Netherlands (2002)
- Lindenmayer, A.: Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology* 18(3), 300–315 (1968)
- Littman, M.L., Dean, T.L., Kaelbling, L.P.: On the complexity of solving Markov decision processes. In: Proceedings of the Eleventh International Conference on Uncertainty in Artificial Intelligence, pp. 394–402 (1995)
- Lucas, S.M., Runarsson, T.P.: Temporal difference learning versus co-evolution for acquiring othello position evaluation. In: IEEE Symposium on Computational Intelligence and Games (2006)
- Lucas, S.M., Togelius, J.: Point-to-point car racing: an initial study of evolution versus temporal difference learning. In: Symposium, I.E.E.E. (ed.) on Computational Intelligence and Games, pp. 260–267 (2007)
- Mahadevan, S., Maggioni, M.: Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research* 8, 2169–2231 (2007)
- Mahfoud, S.: A comparison of parallel and sequential niching methods. In: Conference on Genetic Algorithms, vol. 136, p. 143 (1995)
- McQuesten, P., Miikkulainen, R.: Culling and teaching in neuro-evolution. In: Proceedings of the Seventh International Conference on Genetic Algorithms, pp. 760–767 (1997)
- Meyer, J., Husbands, P., Harvey, I.: Evolutionary robotics: A survey of applications and problems. In: Evolutionary Robotics, pp. 1–21. Springer, Heidelberg (1998)
- Millán, J., Posenato, D., Dedieu, E.: Continuous-action Q-learning. *Machine Learning* 49(2), 247–265 (2002)
- Monroy, G., Stanley, K., Miikkulainen, R.: Coevolution of neural networks using a layered Pareto archive. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, p. 336 (2006)
- Moriarty, D., Miikkulainen, R.: Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation* 5(4), 373–399 (1997)
- Moriarty, D.E., Miikkulainen, R.: Efficient reinforcement learning through symbiotic evolution. *Machine Learning* 22(11), 11–33 (1996)
- Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research* 11, 199–229 (1999)
- Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E.: Inverted autonomous helicopter flight via reinforcement learning. In: Proceedings of the International Symposium on Experimental Robotics (2004)
- Nolfi, S., Parisi, D.: Learning to adapt to changing environments in evolving neural networks. *Adaptive Behavior* 5(1), 75–98 (1997)
- Nolfi, S., Elman, J.L., Parisi, D.: Learning and evolution in neural networks. *Adaptive Behavior* 2, 5–28 (1994)
- Nordin, P., Banzhaf, W.: An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior* 5(2), 107 (1997)
- Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems* 11(3), 387–434 (2005)

- Panait, L., Luke, S., Harrison, J.F.: Archive-based cooperative coevolutionary algorithms. In: GECCO 2006: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 345–352 (2006)
- Parr, R., Painter-Wakefield, C., Li, L., Littman, M.: Analyzing feature generation for value-function approximation. In: Proceedings of the 24th International Conference on Machine Learning, p. 744 (2007)
- Pereira, F.B., Costa, E.: Understanding the role of learning in the evolution of busy beaver: A comparison between the Baldwin Effect and a Lamarckian strategy. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2001 (2001)
- Peters, J., Schaal, S.: Natural actor-critic. *Neurocomputing* 71(7-9), 1180–1190 (2008)
- Pollack, J., Blair, A.: Co-evolution in the successful learning of backgammon strategy. *Machine Learning* 32(3), 225–240 (1998)
- Popovici, E., Bucci, A., Wiegand, P., De Jong, E.: Coevolutionary principles. In: Rozenberg, G., Baecq, T., Kok, J. (eds.) *Handbook of Natural Computing*. Springer, Berlin (2010)
- Potter, M.A., De Jong, K.A.: Evolving neural networks with collaborative species. In: Summer Computer Simulation Conference, pp. 340–345 (1995)
- Potter, M.A., De Jong, K.A.: Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation* 8, 1–29 (2000)
- Pratihar, D.: Evolutionary robotics: A review. *Sadhana* 28(6), 999–1009 (2003)
- Priesterjahn, S., Weimer, A., Eberling, M.: Real-time imitation-based adaptation of gaming behaviour in modern computer games. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1431–1432 (2008)
- Radcliffe, N.: Genetic set recombination and its application to neural network topology optimisation. *Neural Computing & Applications* 1(1), 67–90 (1993)
- Rosin, C.D., Belew, R.K.: New methods for competitive coevolution. *Evolutionary Computation* 5(1), 1–29 (1997)
- Rubinstein, R., Kroese, D.: The cross-entropy method: a unified approach to combinatorial optimization. In: Monte-Carlo Simulation, and Machine Learning. Springer, Heidelberg (2004)
- Runarsson, T.P., Lucas, S.M.: Co-evolution versus self-play temporal difference learning for acquiring position evaluation in small-board go. *IEEE Transactions on Evolutionary Computation* 9, 628–640 (2005)
- Schmidhuber, J., Wierstra, D., Gomez, F.J.: Evolino: Hybrid neuroevolution / optimal linear search for sequence learning. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, pp. 853–858 (2005)
- Schmidhuber, J., Wierstra, D., Gagliolo, M., Gomez, F.: Training recurrent networks by evolino. *Neural Computation* 19(3), 757–779 (2007)
- Schroder, P., Green, B., Grum, N., Fleming, P.: On-line evolution of robust control systems: an industrial active magnetic bearing application. *Control Engineering Practice* 9(1), 37–49 (2001)
- Sigaud, O., Butz, M., Kozlova, O., Meyer, C.: Anticipatory Learning Classifier Systems and Factored Reinforcement Learning. *Anticipatory Behavior in Adaptive Learning Systems*, 321–333 (2009)
- Stanley, K., Miikkulainen, R.: A taxonomy for artificial embryogeny. *Artificial Life* 9(2), 93–130 (2003)
- Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10(2), 99–127 (2002)
- Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research* 21, 63–100 (2004a)

- Stanley, K.O., Miikkulainen, R.: Evolving a Roving Eye for Go. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3103, pp. 1226–1238. Springer, Heidelberg (2004b)
- Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Evolving adaptive neural networks with and without adaptive synapses. In: Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003), vol. 4, pp. 2557–2564 (2003)
- Stanley, K.O., D’Ambrosio, D.B., Gauci, J.: A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life* 15(2), 185–212 (2009)
- Steels, L.: Emergent functionality in robotic agents through on-line evolution. In: Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, pp. 8–16 (1994)
- Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings of the Seventh International Conference on Machine Learning, pp. 216–224 (1990)
- Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
- Sywerda, G.: Uniform crossover in genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms, pp. 2–9 (1989)
- Tan, C., Ang, J., Tan, K., Tay, A.: Online adaptive controller for simulated car racing. In: Congress on Evolutionary Computation (CEC), pp. 2239–2245 (2008)
- Taylor, M.E., Whiteson, S., Stone, P.: Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In: GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1321–1328 (2006)
- Tesauro, G.: TD-gammon, a self-teaching backgammon program achieves master-level play. *Neural Computation* 6, 215–219 (1994)
- Tesauro, G.: Comments on ‘‘co-evolution in the successful learning of backgammon strategy’’. *Machine Learning* 32(3), 241–243 (1998)
- Verbancsics, P., Stanley, K.: Evolving Static Representations for Task Transfer. *Journal of Machine Learning Research* 11, 1737–1769 (2010)
- Von Neumann, J.: Zur Theorie der Gesellschaftsspiele Math. Annalen 100, 295–320 (1928)
- Whiteson, S., Stone, P.: Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research* 7, 877–917 (2006a)
- Whiteson, S., Stone, P.: On-line evolutionary computation for reinforcement learning in stochastic domains. In: GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1577–1584 (2006b)
- Whiteson, S., Kohl, N., Miikkulainen, R., Stone, P.: Evolving keepaway soccer players through task decomposition. *Machine Learning* 59(1), 5–30 (2005)
- Whiteson, S., Tanner, B., White, A.: The reinforcement learning competitions. *AI Magazine* 31(2), 81–94 (2010a)
- Whiteson, S., Taylor, M.E., Stone, P.: Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. *Autonomous Agents and Multi-Agent Systems* 21(1), 1–27 (2010b)
- Whitley, D., Dominic, S., Das, R., Anderson, C.W.: Genetic reinforcement learning for neurocontrol problems. *Machine Learning* 13, 259–284 (1993)
- Whitley, D., Gordon, S., Mathias, K.: Lamarckian evolution, the Baldwin effect and function optimization. In: Parallel Problem Solving from Nature - PPSN III, pp. 6–15 (1994)
- Wiegand, R., Liles, W., De Jong, K.: An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 1235–1242 (2001)
- Wieland, A.: Evolving neural network controllers for unstable systems. In: International Joint Conference on Neural Networks, vol 2, pp. 667–673 (1991)

- Wilson, S.: Classifier fitness based on accuracy. *Evolutionary Computation* 3(2), 149–175 (1995)
- Wilson, S.: Function approximation with a classifier system. In: GECCO 2001: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 974–982 (2001)
- Wolpert, D., Tumer, K.: Optimal payoff functions for members of collectives. *Modeling Complexity in Economic and Social Systems*, 355 (2002)
- Yamasaki, K., Sekiguchi, M.: Clear explanation of different adaptive behaviors between Darwinian population and Lamarckian population in changing environment. In: Proceedings of the Fifth International Symposium on Artificial Life and Robotics, vol. 1, pp. 120–123 (2000)
- Yao, X.: Evolving artificial neural networks. *Proceedings of the IEEE* 87(9), 1423–1447 (1999)
- Yong, C.H., Miikkulainen, R.: Coevolution of role-based cooperation in multi-agent systems. Tech. Rep. AI07-338, Department of Computer Sciences, The University of Texas at Austin (2007)
- Zhang, B., Muhlenbein, H.: Evolving optimal neural networks using genetic algorithms with Occam's razor. *Complex Systems* 7(3), 199–220 (1993)
- Zufferey, J.-C., Floreano, D., van Leeuwen, M., Merenda, T.: Evolving vision-based flying robots. In: Bühlhoff, H.H., Lee, S.-W., Poggio, T.A., Wallraven, C. (eds.) *BMCV 2002. LNCS*, vol. 2525, pp. 592–600. Springer, Heidelberg (2002)

Part IV
Probabilistic Models of Self and Others

Chapter 11

Bayesian Reinforcement Learning

Nikos Vlassis, Mohammad Ghavamzadeh, Shie Mannor, and Pascal Poupart

Abstract. This chapter surveys recent lines of work that use Bayesian techniques for reinforcement learning. In Bayesian learning, uncertainty is expressed by a prior distribution over unknown parameters and learning is achieved by computing a posterior distribution based on the data observed. Hence, Bayesian reinforcement learning distinguishes itself from other forms of reinforcement learning by explicitly maintaining a distribution over various quantities such as the parameters of the model, the value function, the policy or its gradient. This yields several benefits: a) domain knowledge can be naturally encoded in the prior distribution to speed up learning; b) the exploration/exploitation tradeoff can be naturally optimized; and c) notions of risk can be naturally taken into account to obtain robust policies.

11.1 Introduction

Bayesian reinforcement learning is perhaps the oldest form of reinforcement learning. Already in the 1950's and 1960's, several researchers in Operations Research

Nikos Vlassis
(1) Luxembourg Centre for Systems Biomedicine, University of Luxembourg, and
(2) OneTree Financials, Luxembourg
e-mail: nikos.vlassis@uni.lu, nikos@onetreefinancials.com

Mohammad Ghavamzadeh
INRIA
e-mail: mohammad.ghavamzadeh@inria.fr

Shie Mannor
Technion
e-mail: shie@ee.technion.ac.il

Pascal Poupart
University of Waterloo
e-mail: ppoupart@cs.uwaterloo.ca

studied the problem of controlling Markov chains with uncertain probabilities. Bellman developed dynamic programming techniques for Bayesian bandit problems (Bellman, 1956; Bellman and Kalaba, 1959; Bellman, 1961). This work was then generalized to multi-state sequential decision problems with unknown transition probabilities and rewards (Silver, 1963; Cozzolino, 1964; Cozzolino et al, 1965). The book “Bayesian Decision Problems and Markov Chains” by Martin (1967) gives a good overview of the work of that era. At the time, reinforcement learning was known as *adaptive control processes* and then *Bayesian adaptive control*.

Since Bayesian learning meshes well with decision theory, Bayesian techniques are natural candidates to simultaneously learn about the environment while making decisions. The idea is to treat the unknown parameters as random variables and to maintain an explicit distribution over these variables to quantify the uncertainty. As evidence is gathered, this distribution is updated and decisions can be made simply by integrating out the unknown parameters.

In contrast to traditional reinforcement learning techniques that typically learn point estimates of the parameters, the use of an explicit distribution permits a quantification of the uncertainty that can speed up learning and reduce risk. In particular, the prior distribution allows the practitioner to encode domain knowledge that can reduce the uncertainty. For most real-world problems, reinforcement learning from scratch is intractable since too many parameters would have to be learned if the transition, observation and reward functions are completely unknown. Hence, by encoding domain knowledge in the prior distribution, the amount of interaction with the environment to find a good policy can be reduced significantly. Furthermore, domain knowledge can help avoid catastrophic events that would have to be learned by repeated trials otherwise. An explicit distribution over the parameters also provides a quantification of the uncertainty that is very useful to optimize the exploration/exploitation tradeoff. The choice of action is typically done to maximize future rewards based on the current estimate of the model (exploitation), however there is also a need to explore the uncertain parts of the model in order to refine it and earn higher rewards in the future. Hence, the quantification of this uncertainty by an explicit distribution becomes very useful. Similarly, an explicit quantification of the uncertainty of the future returns can be used to minimize variance or the risk of low rewards.

The chapter is organized as follows. Section 11.2 describes Bayesian techniques for *model-free* reinforcement learning where explicit distributions over the parameters of the value function, the policy or its gradient are maintained. Section 11.3 describes Bayesian techniques for *model-based* reinforcement learning, where the distributions are over the parameters of the transition, observation and reward functions. Finally, Section 11.4 describes Bayesian techniques that take into account the availability of finitely many samples to obtain sample complexity bounds and for optimization under uncertainty.

11.2 Model-Free Bayesian Reinforcement Learning

Model-free RL methods are those that do not explicitly learn a model of the system and only use sample trajectories obtained by direct interaction with the system. Model-free techniques are often simpler to implement since they do not require any data structure to represent a model nor any algorithm to update this model. However, it is often more complicated to reason about model-free approaches since it is not always obvious how sample trajectories should be used to update an estimate of the optimal policy or value function. In this section, we describe several Bayesian techniques that treat the value function or policy gradient as random objects drawn from a distribution. More specifically, Section 11.2.1 describes approaches to learn distributions over Q-functions, Section 11.2.2 considers distributions over policy gradients and Section 11.2.3 shows how distributions over value functions can be used to infer distributions over policy gradients in actor-critic algorithms.

11.2.1 Value-Function Based Algorithms

Value-function based RL methods search in the space of value functions to find the optimal value (action-value) function, and then use it to extract an optimal policy. In this section, we study two Bayesian value-function based RL algorithms: Bayesian Q-learning (Dearden et al, 1998) and Gaussian process temporal difference learning (Engel et al, 2003, 2005a; Engel, 2005). The first algorithm caters to domains with discrete state and action spaces while the second algorithm handles continuous state and action spaces.

11.2.1.1 Bayesian Q-Learning

Bayesian Q-learning (BQL) (Dearden et al, 1998) is a Bayesian approach to the widely-used Q-learning algorithm (Watkins, 1989), in which exploration and exploitation are balanced by explicitly maintaining a distribution over Q-values to help select actions. Let $D(s,a)$ be a random variable that denotes the sum of discounted rewards received when action a is taken in state s and an optimal policy is followed thereafter. The expectation of this variable $\mathbb{E}[D(s,a)] = Q(s,a)$ is the classic Q-function. In BQL, we place a prior over $D(s,a)$ for any state $s \in \mathcal{S}$ and any action $a \in \mathcal{A}$, and update its posterior when we observe independent samples of $D(s,a)$. The goal in BQL is to learn $Q(s,a)$ by reducing the uncertainty about $\mathbb{E}[D(s,a)]$. BQL makes the following simplifying assumptions: (1) Each $D(s,a)$ follows a normal distribution with mean $\mu(s,a)$ and precision $\tau(s,a)$.¹ This assumption implies that to model our uncertainty about the distribution of $D(s,a)$, it suffices to model a distribution over $\mu(s,a)$ and $\tau(s,a)$. (2) The prior $P(D(s,a))$ for each (s,a) -pair is assumed to be independent and normal-Gamma distributed. This assumption restricts the form of prior knowledge about the system, but ensures that the

¹ The precision of a Gaussian random variable is the inverse of its variance.

posterior $P(D(s,a)|d)$ given a sampled sum of discounted rewards $d = \sum_t \gamma^t r(s_t, a_t)$ is also normal-Gamma distributed. However, since the sum of discounted rewards for different (s,a) -pairs are related by Bellman's equation, the posterior distributions become correlated. (3) To keep the representation simple, the posterior distributions are forced to be independent by breaking the correlations.

In BQL, instead of storing the Q-values as in standard Q-learning, we store the hyper-parameters of the distributions over each $D(s,a)$. Therefore, BQL, in its original form, can only be applied to MDPs with finite state and action spaces. At each time step, after executing a in s and observing r and s' , the distributions over the D 's are updated as follows:

$$\begin{aligned} P(D(s,a)|r,s') &= \int_d P(D(s,a)|r + \gamma d) P(D(s',a') = d) \\ &\propto \int_d P(D(s,a)) P(r + \gamma d|D(s,a)) P(D(s',a') = d) \end{aligned}$$

Since the posterior does not have a closed form due to the integral, it is approximated by finding the closest Normal-Gamma distribution by minimizing KL-divergence.

At run-time, it is very tempting to select the action with the highest expected Q-value (i.e., $a^* = \arg \max_a \mathbb{E}[Q(s,a)]$), however this strategy does not ensure exploration. To address this, Dearden et al (1998) proposed to add an exploration bonus to the expected Q-values that estimates the myopic *value of perfect information* (VPI).

$$a^* = \arg \max_a \mathbb{E}[Q(s,a)] + VPI(s,a)$$

If exploration leads to a policy change, then the gain in value should be taken into account. Since the agent does not know in advance the effect of each action, VPI is computed as an expected gain

$$VPI(s,a) = \int_{-\infty}^{\infty} dx \text{Gain}_{s,a}(x) P(Q(s,a) = x) \quad (11.1)$$

where the gain corresponds to the improvement induced by learning the *exact* Q-value (denoted by $q_{s,a}$) of the action executed.

$$\text{Gain}_{s,a}(q_{s,a}) = \begin{cases} q_{s,a} - \mathbb{E}[Q(s,a_1)] & \text{if } a \neq a_1 \text{ and } q_{s,a} > \mathbb{E}[Q(s,a_1)] \\ \mathbb{E}[Q(s,a_2)] - q_{s,a} & \text{if } a = a_1 \text{ and } q_{s,a} < \mathbb{E}[Q(s,a_2)] \\ 0 & \text{otherwise} \end{cases} \quad (11.2)$$

There are two cases: a is revealed to have a higher Q-value than the action a_1 with the highest expected Q-value or the action a_1 with the highest expected Q-value is revealed to have a lower Q-value than the action a_2 with the second highest expected Q-value.

11.2.1.2 Gaussian Process Temporal Difference Learning

Bayesian Q-learning (BQL) maintains a separate distribution over $D(s,a)$ for each (s,a) -pair, thus, it cannot be used for problems with continuous state or action

spaces. Engel et al (2003, 2005a) proposed a natural extension that uses Gaussian processes. As in BQL, $D(s,a)$ is assumed to be Normal with mean $\mu(s,a)$ and precision $\tau(s,a)$. However, instead of maintaining a Normal-Gamma over μ and τ simultaneously, a Gaussian over μ is modeled. Since $\mu(s,a) = Q(s,a)$ and the main quantity that we want to learn is the Q-function, it would be fine to maintain a belief only about the mean. To accommodate infinite state and action spaces, a *Gaussian process* is used to model infinitely many Gaussians over $Q(s,a)$ for each (s,a) -pair.

A Gaussian process (e.g., Rasmussen and Williams 2006) is the extension of the multivariate Gaussian distribution to infinitely many dimensions or equivalently, corresponds to infinitely many correlated univariate Gaussians. Gaussian processes $GP(\mu,k)$ are parameterized by a mean function $\mu(x)$ and a kernel function $k(x,x')$ which are the limit of the mean vector and covariance matrix of multivariate Gaussians when the number of dimensions become infinite. Gaussian processes are often used for functional regression based on sampled realizations of some unknown underlying function.

Along those lines, Engel et al (2003, 2005a) proposed a *Gaussian Process Temporal Difference* (GPTD) approach to learn the Q-function of a policy based on samples of discounted sums of returns. Recall that the distribution of the sum of discounted rewards for a fixed policy π is defined recursively as follows:

$$D(\mathbf{z}) = r(\mathbf{z}) + \gamma D(\mathbf{z}') \quad \text{where } \mathbf{z}' \sim P^\pi(\mathbf{z}'|\mathbf{z}). \quad (11.3)$$

When \mathbf{z} refers to states then $\mathbb{E}[D] = V$ and when it refers to state-action pairs then $\mathbb{E}[D] = Q$. Unless otherwise specified, we will assume that $\mathbf{z} = (s,a)$. We can decompose D as the sum of its mean Q and a zero-mean noise term ΔQ , which will allow us to place a distribution directly over Q later on. Replacing $D(\mathbf{z})$ by $Q(\mathbf{z}) + \Delta Q(\mathbf{z})$ in Eq. 11.3 and grouping the ΔQ terms into a single zero-mean noise term $N(\mathbf{z},\mathbf{z}') = \Delta Q(\mathbf{z}) - \gamma \Delta Q(\mathbf{z}')$, we obtain

$$r(\mathbf{z}) = Q(\mathbf{z}) - \gamma Q(\mathbf{z}') + N(\mathbf{z},\mathbf{z}') \quad \text{where } \mathbf{z}' \sim P^\pi(\mathbf{z}'|\mathbf{z}). \quad (11.4)$$

The GPTD learning model (Engel et al, 2003, 2005a) is based on the statistical generative model in Eq. 11.4 that relates the observed reward signal r to the unobserved action-value function Q . Now suppose that we observe the sequence $\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_t$, then Eq. 11.4 leads to a system of t equations that can be expressed in matrix form as

$$r_{t-1} = H_t Q_t + N_t, \quad (11.5)$$

where

$$\begin{aligned} r_t &= (r(\mathbf{z}_0), \dots, r(\mathbf{z}_t))^\top, & Q_t &= (Q(\mathbf{z}_0), \dots, Q(\mathbf{z}_t))^\top, \\ N_t &= (N(\mathbf{z}_0, \mathbf{z}_1), \dots, N(\mathbf{z}_{t-1}, \mathbf{z}_t))^\top, \end{aligned} \quad (11.6)$$

$$H_t = \begin{bmatrix} 1-\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & -\gamma \end{bmatrix}. \quad (11.7)$$

If we assume that the residuals $\Delta Q(\mathbf{z}_0), \dots, \Delta Q(\mathbf{z}_t)$ are zero-mean Gaussians with variance σ^2 , and moreover, each residual is generated independently of all the others, i.e., $\mathbb{E}[\Delta Q(\mathbf{z}_i)\Delta Q(\mathbf{z}_j)] = 0$, for $i \neq j$, it is easy to show that the noise vector N_t is Gaussian with mean 0 and the covariance matrix

$$\Sigma_t = \sigma^2 H_t H_t^\top = \sigma^2 \begin{bmatrix} 1+\gamma^2 & -\gamma & 0 & \dots & 0 \\ -\gamma & 1+\gamma^2 & -\gamma & \dots & 0 \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \dots & -\gamma & 1+\gamma^2 \end{bmatrix}. \quad (11.8)$$

In episodic tasks, if \mathbf{z}_{t-1} is the last state-action pair in the episode (i.e., s_t is a zero-reward absorbing terminal state), H_t becomes a square $t \times t$ invertible matrix of the form shown in Eq. 11.7 with its last column removed. The effect on the noise covariance matrix Σ_t is that the bottom-right element becomes 1 instead of $1 + \gamma^2$.

Placing a GP prior $GP(0, k)$ on Q , we may use Bayes' rule to obtain the moments \hat{Q} and \hat{k} of the posterior Gaussian process on Q :

$$\begin{aligned} \hat{Q}_t(\mathbf{z}) &= \mathbb{E}[Q(\mathbf{z})|\mathcal{D}_t] = k_t(\mathbf{z})^\top \alpha_t, \\ \hat{k}_t(\mathbf{z}, \mathbf{z}') &= \mathbf{Cov}[Q(\mathbf{z}), Q(\mathbf{z}')|\mathcal{D}_t] = k(\mathbf{z}, \mathbf{z}') - k_t(\mathbf{z})^\top C_t k_t(\mathbf{z}'), \end{aligned} \quad (11.9)$$

where \mathcal{D}_t denotes the observed data up to and including time step t . We used here the following definitions:

$$\begin{aligned} k_t(\mathbf{z}) &= (k(\mathbf{z}_0, \mathbf{z}), \dots, k(\mathbf{z}_t, \mathbf{z}))^\top, & K_t &= [k_t(\mathbf{z}_0), k_t(\mathbf{z}_1), \dots, k_t(\mathbf{z}_t)], \\ \alpha_t &= H_t^\top (H_t K_t H_t^\top + \Sigma_t)^{-1} r_{t-1}, & C_t &= H_t^\top (H_t K_t H_t^\top + \Sigma_t)^{-1} H_t. \end{aligned} \quad (11.10)$$

As more samples are observed, the posterior covariance decreases, reflecting a growing confidence in the Q-function estimate \hat{Q}_t .

The GPTD model described above is kernel-based and non-parametric. It is also possible to employ a parametric representation under very similar assumptions. In the parametric setting, the GP Q is assumed to consist of a linear combination of a finite number of basis functions: $Q(\cdot, \cdot) = \phi(\cdot, \cdot)^\top W$, where ϕ is the feature vector and W is the weight vector. In the parametric GPTD, the randomness in Q is due to W being a random vector. In this model, we place a Gaussian prior over W and apply Bayes' rule to calculate the posterior distribution of W conditioned on the observed data. The posterior mean and covariance of Q may be easily computed by multiplying the posterior moments of W with the feature vector ϕ . See Engel (2005) for more details on parametric GPTD.

In the parametric case, the computation of the posterior may be performed online in $O(n^2)$ time per sample and $O(n^2)$ memory, where n is the number of basis functions used to approximate Q . In the non-parametric case, we have a new basis function for each new sample we observe, making the cost of adding the t 'th sample $O(t^2)$ in both time and memory. This would seem to make the non-parametric form of GPTD computationally infeasible except in small and simple problems. However, the computational cost of non-parametric GPTD can be reduced by using an online sparsification method (e.g., Engel et al 2002), to a level that it can be efficiently implemented online.

The choice of the prior distribution may significantly affect the performance of GPTD. However, in the standard GPTD, the prior is set at the beginning and remains unchanged during the execution of the algorithm. Reisinger et al (2008) developed an online model selection method for GPTD using sequential MC techniques, called *replacing-kernel RL*, and empirically showed that it yields better performance than the standard GPTD for many different kernel families.

Finally, the GPTD model can be used to derive a SARSA-type algorithm, called GPSARSA (Engel et al, 2005a; Engel, 2005), in which state-action values are estimated using GPTD and policies are improved by a ϵ -greedily strategy while slowly decreasing ϵ toward 0. The GPTD framework, especially the GPSARSA algorithm, has been successfully applied to large scale RL problems such as the control of an octopus arm (Engel et al, 2005b) and wireless network association control (Aharony et al, 2005).

11.2.2 Policy Gradient Algorithms

Policy gradient (PG) methods are RL algorithms that maintain a parameterized action-selection policy and update the policy parameters by moving them in the direction of an estimate of the gradient of a performance measure (e.g., Williams 1992; Marbach 1998; Baxter and Bartlett 2001). These algorithms have been theoretically and empirically analyzed (e.g., Marbach 1998; Baxter and Bartlett 2001), and also extended to POMDPs (Baxter and Bartlett, 2001). However, both the theoretical results and empirical evaluations have highlighted a major shortcoming of these algorithms, namely, the high variance of the gradient estimates.

Several solutions have been proposed for this problem such as: (1) To use an artificial *discount factor* ($0 < \gamma < 1$) in these algorithms (Marbach, 1998; Baxter and Bartlett, 2001). However, this creates another problem by introducing bias into the gradient estimates. (2) To subtract a *reinforcement baseline* from the average reward estimate in the updates of PG algorithms (Williams, 1992; Marbach, 1998; Sutton et al, 2000; Greensmith et al, 2004). This approach does not involve biasing the gradient estimate, however, what would be a good choice for a state-dependent baseline is more or less an open question. (3) To replace the policy gradient estimate with an estimate of the so-called *natural* policy gradient (Kakade, 2002; Bagnell and Schneider, 2003; Peters et al, 2003). In terms of the policy update rule, the move to a

natural-gradient rule amounts to linearly transforming the gradient using the inverse Fisher information matrix of the policy. In empirical evaluations, natural PG has been shown to significantly outperform conventional PG (Kakade, 2002; Bagnell and Schneider, 2003; Peters et al, 2003; Peters and Schaal, 2008).

However, both conventional and natural policy gradient methods rely on Monte-Carlo (MC) techniques in estimating the gradient of the performance measure. Although MC estimates are unbiased, they tend to suffer from high variance, or alternatively, require excessive sample sizes (see O'Hagan, 1987 for a discussion). In the case of policy gradient estimation this is exacerbated by the fact that consistent policy improvement requires multiple gradient estimation steps. O'Hagan (1991) proposes a Bayesian alternative to MC estimation of an integral, called *Bayesian quadrature* (BQ). The idea is to model integrals of the form $\int dx f(x)g(x)$ as random quantities. This is done by treating the first term in the integrand, f , as a random function over which we express a prior in the form of a Gaussian process (GP). Observing (possibly noisy) samples of f at a set of points $\{x_1, x_2, \dots, x_M\}$ allows us to employ Bayes' rule to compute a posterior distribution of f conditioned on these samples. This, in turn, induces a posterior distribution over the value of the integral. Rasmussen and Ghahramani (2003) experimentally demonstrated how this approach, when applied to the evaluation of an expectation, can outperform MC estimation by orders of magnitude, in terms of the mean-squared error. Interestingly, BQ is often effective even when f is known. The posterior of f can be viewed as an approximation of f (that converges to f in the limit), but this approximation can be used to perform the integration in closed form. In contrast, MC integration uses the exact f , but only at the points sampled. So BQ makes better use of the information provided by the samples by using the posterior to "interpolate" between the samples and by performing the integration in closed form.

In this section, we study a Bayesian framework for policy gradient estimation based on modeling the policy gradient as a GP (Ghavamzadeh and Engel, 2006). This reduces the number of samples needed to obtain accurate gradient estimates. Moreover, estimates of the natural gradient as well as a measure of the uncertainty in the gradient estimates, namely, the gradient covariance, are provided at little extra cost.

Let us begin with some definitions and notations. A *stationary policy* $\pi(\cdot|s)$ is a probability distribution over actions, conditioned on the current state. Given a fixed policy π , the MDP induces a Markov chain over state-action pairs, whose transition probability from (s_t, a_t) to (s_{t+1}, a_{t+1}) is $\pi(a_{t+1}|s_{t+1})P(s_{t+1}|s_t, a_t)$. We generically denote by $\xi = (s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T)$, $T \in \{0, 1, \dots, \infty\}$ a path generated by this Markov chain. The probability (density) of such a path is given by

$$P(\xi|\pi) = P_0(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t)P(s_{t+1}|s_t, a_t). \quad (11.11)$$

We denote by $R(\xi) = \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)$ the discounted *cumulative return* of the path ξ , where $\gamma \in [0, 1]$ is a discount factor. $R(\xi)$ is a random variable both because the path ξ itself is a random variable, and because, even for a given path, each of the

rewards sampled in it may be stochastic. The expected value of $R(\xi)$ for a given path ξ is denoted by $\bar{R}(\xi)$. Finally, we define the *expected return* of policy π as

$$\eta(\pi) = \mathbb{E}[R(\xi)] = \int d\xi \bar{R}(\xi)P(\xi|\pi). \quad (11.12)$$

In PG methods, we define a class of smoothly parameterized stochastic policies $\{\pi(\cdot|s; \theta), s \in \mathcal{S}, \theta \in \Theta\}$. We estimate the gradient of the expected return w.r.t. the policy parameters θ , from the observed system trajectories. We then improve the policy by adjusting the parameters in the direction of the gradient. We use the following equation to estimate the gradient of the expected return:

$$\nabla \eta(\theta) = \int d\xi \bar{R}(\xi) \frac{\nabla P(\xi; \theta)}{P(\xi; \theta)} P(\xi; \theta), \quad (11.13)$$

where $\frac{\nabla P(\xi; \theta)}{P(\xi; \theta)} = \nabla \log P(\xi; \theta)$ is called the *score function* or *likelihood ratio*. Since the initial-state distribution P_0 and the state-transition distribution P are independent of the policy parameters θ , we may write the score function of a path ξ using Eq. 11.11 as²

$$u(\xi; \theta) = \frac{\nabla P(\xi; \theta)}{P(\xi; \theta)} = \sum_{t=0}^{T-1} \frac{\nabla \pi(a_t|s_t; \theta)}{\pi(a_t|s_t; \theta)} = \sum_{t=0}^{T-1} \nabla \log \pi(a_t|s_t; \theta). \quad (11.14)$$

The frequentist approach to PG uses classical MC to estimate the gradient in Eq. 11.13. This method generates i.i.d. sample paths ξ_1, \dots, ξ_M according to $P(\xi; \theta)$, and estimates the gradient $\nabla \eta(\theta)$ using the MC estimator

$$\widehat{\nabla \eta}(\theta) = \frac{1}{M} \sum_{i=1}^M R(\xi_i) \nabla \log P(\xi_i; \theta) = \frac{1}{M} \sum_{i=1}^M R(\xi_i) \sum_{t=0}^{T_i-1} \nabla \log \pi(a_{t,i}|s_{t,i}; \theta). \quad (11.15)$$

This is an unbiased estimate, and therefore, by the law of large numbers, $\widehat{\nabla \eta}(\theta) \rightarrow \nabla \eta(\theta)$ as M goes to infinity, with probability one.

In the frequentist approach to PG, the performance measure used is $\eta(\theta)$. In order to serve as a useful performance measure, it has to be a deterministic function of the policy parameters θ . This is achieved by averaging the cumulative return $R(\xi)$ over all possible paths ξ and all possible returns accumulated in each path. In the Bayesian approach we have an additional source of randomness, namely, our subjective Bayesian uncertainty concerning the process generating the cumulative return. Let us denote $\eta_B(\theta) = \int d\xi \bar{R}(\xi)P(\xi; \theta)$, where $\eta_B(\theta)$ is a random variable because of the Bayesian uncertainty. We are interested in evaluating the posterior distribution of the *gradient* of $\eta_B(\theta)$ w.r.t. the policy parameters θ . The posterior mean of the gradient is

² To simplify notation, we omit ∇ and u 's dependence on the policy parameters θ , and use ∇ and $u(\xi)$ in place of ∇_θ and $u(\xi; \theta)$ in the sequel.

$$\mathbb{E}[\nabla \eta_B(\theta) | \mathcal{D}_M] = \mathbb{E}\left[\int d\xi R(\xi) \frac{\nabla P(\xi; \theta)}{P(\xi; \theta)} P(\xi; \theta) \Big| \mathcal{D}_M\right]. \quad (11.16)$$

In the Bayesian policy gradient (BPG) method of Ghavamzadeh and Engel (2006), the problem of estimating the gradient of the expected return (Eq. 11.16) is cast as an integral evaluation problem, and then the BQ method (O'Hagan, 1991), described above, is used. In BQ, we need to partition the integrand into two parts, $f(\xi; \theta)$ and $g(\xi; \theta)$. We will model f as a GP and assume that g is a function known to us. We will then proceed by calculating the posterior moments of the gradient $\nabla \eta_B(\theta)$ conditioned on the observed data $\mathcal{D}_M = \{\xi_1, \dots, \xi_M\}$. Because in general, $R(\xi)$ cannot be known exactly, even for a given ξ (due to the stochasticity of the rewards), $R(\xi)$ should always belong to the GP part of the model, i.e., $f(\xi; \theta)$. Ghavamzadeh and Engel (2006) proposed two different ways of partitioning the integrand in Eq. 11.16, resulting in two distinct Bayesian models. Table 1 in Ghavamzadeh and Engel (2006) summarizes the two models. Models 1 and 2 use Fisher-type kernels for the prior covariance of f . The choice of Fisher-type kernels was motivated by the notion that a good representation should depend on the data generating process (see Jaakkola and Haussler 1999; Shawe-Taylor and Cristianini 2004 for a thorough discussion). The particular choices of linear and quadratic Fisher kernels were guided by the requirement that the posterior moments of the gradient be analytically tractable.

Models 1 and 2 can be used to define algorithms for evaluating the gradient of the expected return w.r.t. the policy parameters. The algorithm (for either model) takes a set of policy parameters θ and a sample size M as input, and returns an estimate of the posterior moments of the gradient of the expected return. This Bayesian PG evaluation algorithm, in turn, can be used to derive a Bayesian policy gradient (BPG) algorithm that starts with an initial vector of policy parameters θ_0 and updates the parameters in the direction of the posterior mean of the gradient of the expected return, computed by the Bayesian PG evaluation procedure. This is repeated N times, or alternatively, until the gradient estimate is sufficiently close to zero.

As mentioned earlier, the kernel functions used in Models 1 and 2 are both based on the Fisher information matrix $G(\theta)$. Consequently, every time we update the policy parameters we need to recompute G . In most practical situations, G is not known and needs to be estimated. Ghavamzadeh and Engel (2006) described two possible approaches to this problem: MC estimation of G and maximum likelihood (ML) estimation of the MDP's dynamics and use it to calculate G . They empirically showed that even when G is estimated using MC or ML, BPG performs better than MC-based PG algorithms.

BPG may be made significantly more efficient, both in time and memory, by sparsifying the solution. Such sparsification may be performed incrementally, and helps to numerically stabilize the algorithm when the kernel matrix is singular, or nearly so. Similar to the GPTD case, one possibility is to use the on-line sparsification method proposed by Engel et al (2002) to selectively add a new observed path to a set of *dictionary* paths, which are used as a basis for approximating the

full solution. Finally, it is easy to show that the BPG models and algorithms can be extended to POMDPs along the same lines as in Baxter and Bartlett (2001).

11.2.3 Actor-Critic Algorithms

Actor-critic (AC) methods were among the earliest to be investigated in RL (Barto et al, 1983; Sutton, 1984). They comprise a family of RL methods that maintain two distinct algorithmic components: an *actor*, whose role is to maintain and update an action-selection policy; and a *critic*, whose role is to estimate the value function associated with the actor’s policy. A common practice is that the actor updates the policy parameters using stochastic gradient ascent, and the critic estimates the value function using some form of temporal difference (TD) learning (Sutton, 1988). When the representations used for the actor and the critic are *compatible*, in the sense explained in Sutton et al (2000) and Konda and Tsitsiklis (2000), the resulting AC algorithm is simple, elegant, and provably convergent (under appropriate conditions) to a local maximum of the performance measure used by the critic plus a measure of the TD error inherent in the function approximation scheme (Konda and Tsitsiklis, 2000; Bhatnagar et al, 2009). The apparent advantage of AC algorithms (e.g., Sutton et al 2000; Konda and Tsitsiklis 2000; Peters et al 2005; Bhatnagar et al 2007) over PG methods, which avoid using a critic, is that using a critic tends to reduce the variance of the policy gradient estimates, making the search in policy-space more efficient and reliable.

Most AC algorithms are based on parametric critics that are updated to optimize frequentist fitness criteria. However, the GPTD model described in Section 11.2.1, provides us with a Bayesian class of critics that return a full posterior distribution over value functions. In this section, we study a Bayesian actor-critic (BAC) algorithm that incorporates GPTD in its critic (Ghavamzadeh and Engel, 2007). We show how the posterior moments returned by the GPTD critic allow us to obtain closed-form expressions for the posterior moments of the policy gradient. This is made possible by utilizing the Fisher kernel (Shawe-Taylor and Cristianini, 2004) as our prior covariance kernel for the GPTD state-action *advantage* values. This is a natural extension of the BPG approach described in Section 11.2.2. It is important to note that while in BPG the basic observable unit, upon which learning and inference are based, is a complete trajectory, BAC takes advantage of the Markov property of the system trajectories and uses individual state-action-reward transitions as its basic observable unit. This helps reduce variance in the gradient estimates, resulting in steeper learning curves compared to BPG and the classic MC approach.

Under certain regularity conditions (Sutton et al, 2000), the expected return of a policy π defined by Eq. 11.12 can be written as

$$\eta(\pi) = \int_{\mathcal{Z}} d\mathbf{z} \mu^\pi(\mathbf{z}) \bar{r}(\mathbf{z}),$$

where $\bar{r}(\mathbf{z})$ is the mean reward for the state-action pair \mathbf{z} , and $\mu^\pi(\mathbf{z}) = \sum_{t=0}^{\infty} \gamma^t P_t^\pi(\mathbf{z})$ is a discounted weighting of state-action pairs encountered while following policy π . Integrating a out of $\mu^\pi(\mathbf{z}) = \mu^\pi(s, a)$ results in the corresponding discounted weighting of states encountered by following policy π ; $\rho^\pi(s) = \int_{\mathcal{A}} da \mu^\pi(s, a)$. Unlike ρ^π and μ^π , $(1 - \gamma)\rho^\pi$ and $(1 - \gamma)\mu^\pi$ are distributions. They are analogous to the stationary distributions over states and state-action pairs of policy π in the undiscounted setting, since as $\gamma \rightarrow 1$, they tend to these stationary distributions, if they exist. The policy gradient theorem (Marbach, 1998, Proposition 1; Sutton et al, 2000, Theorem 1; Konda and Tsitsiklis, 2000, Theorem 1) states that the gradient of the expected return for parameterized policies is given by

$$\nabla \eta(\theta) = \int ds da \rho(s; \theta) \nabla \pi(a|s; \theta) Q(s, a; \theta) = \int d\mathbf{z} \mu(\mathbf{z}; \theta) \nabla \log \pi(a|s; \theta) Q(\mathbf{z}; \theta). \quad (11.17)$$

Observe that if $b : \mathcal{S} \rightarrow \mathbb{R}$ is an arbitrary function of s (also called a *baseline*), then

$$\begin{aligned} \int_{\mathcal{S}} ds da \rho(s; \theta) \nabla \pi(a|s; \theta) b(s) &= \int_{\mathcal{S}} ds \rho(s; \theta) b(s) \nabla \left(\int_{\mathcal{A}} da \pi(a|s; \theta) \right) \\ &= \int_{\mathcal{S}} ds \rho(s; \theta) b(s) \nabla(1) = 0, \end{aligned}$$

and thus, for any baseline $b(s)$, Eq. 11.17 may be written as

$$\nabla \eta(\theta) = \int_{\mathcal{S}} d\mathbf{z} \mu(\mathbf{z}; \theta) \nabla \log \pi(a|s; \theta) [Q(\mathbf{z}; \theta) + b(s)]. \quad (11.18)$$

Now consider the case in which the action-value function for a fixed policy π , Q^π , is approximated by a learned function approximator. If the approximation is sufficiently good, we may hope to use it in place of Q^π in Eqs. 11.17 and 11.18, and still point roughly in the direction of the true gradient. Sutton et al (2000) and Konda and Tsitsiklis (2000) showed that if the approximation $\hat{Q}^\pi(\cdot; \mathbf{w})$ with parameter \mathbf{w} is *compatible*, i.e., $\nabla_{\mathbf{w}} \hat{Q}^\pi(s, a; \mathbf{w}) = \nabla \log \pi(a|s; \theta)$, and if it minimizes the mean squared error

$$\mathcal{E}^\pi(\mathbf{w}) = \int_{\mathcal{S}} d\mathbf{z} \mu^\pi(\mathbf{z}) [Q^\pi(\mathbf{z}) - \hat{Q}^\pi(\mathbf{z}; \mathbf{w})]^2 \quad (11.19)$$

for parameter value \mathbf{w}^* , then we may replace Q^π with $\hat{Q}^\pi(\cdot; \mathbf{w}^*)$ in Eqs. 11.17 and 11.18. An approximation for the action-value function, in terms of a linear combination of basis functions, may be written as $\hat{Q}^\pi(\mathbf{z}; \mathbf{w}) = \mathbf{w}^\top \psi(\mathbf{z})$. This approximation is compatible if the ψ 's are compatible with the policy, i.e., $\psi(\mathbf{z}; \theta) = \nabla \log \pi(a|s; \theta)$. It can be shown that the mean squared-error problems of Eq. 11.19 and

$$\mathcal{E}^\pi(\mathbf{w}) = \int_{\mathcal{S}} d\mathbf{z} \mu^\pi(\mathbf{z}) [Q^\pi(\mathbf{z}) - \mathbf{w}^\top \psi(\mathbf{z}) - b(s)]^2 \quad (11.20)$$

have the same solutions (e.g., Bhatnagar et al 2007, 2009), and if the parameter \mathbf{w} is set to be equal to \mathbf{w}^* in Eq. 11.20, then the resulting mean squared error $\mathcal{E}^\pi(\mathbf{w}^*)$ is further minimized by setting $b(s) = V^\pi(s)$ (Bhatnagar et al, 2007, 2009). In other words, the variance in the action-value function estimator is minimized if the

baseline is chosen to be the value function itself. This means that it is more meaningful to consider $\mathbf{w}^{*\top} \psi(\mathbf{z})$ as the least-squared optimal parametric representation for the *advantage* function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ rather than the action-value function $Q^\pi(s,a)$.

We are now in a position to describe the main idea behind the BAC approach. Making use of the linearity of Eq. 11.17 in Q and denoting $g(\mathbf{z}; \theta) = \mu^\pi(\mathbf{z}) \nabla \log \pi(a|s; \theta)$, we obtain the following expressions for the posterior moments of the policy gradient (O'Hagan, 1991):

$$\begin{aligned}\mathbb{E}[\nabla \eta(\theta)|\mathcal{D}_t] &= \int_{\mathcal{Z}} d\mathbf{z} g(\mathbf{z}; \theta) \hat{Q}_t(\mathbf{z}; \theta) = \int_{\mathcal{Z}} d\mathbf{z} g(\mathbf{z}; \theta) k_t(\mathbf{z})^\top \alpha_t, \\ \mathbf{Cov}[\nabla \eta(\theta)|\mathcal{D}_t] &= \int_{\mathcal{Z}^2} d\mathbf{z} d\mathbf{z}' g(\mathbf{z}; \theta) \hat{S}_t(\mathbf{z}, \mathbf{z}') g(\mathbf{z}'; \theta)^\top \\ &= \int_{\mathcal{Z}^2} d\mathbf{z} d\mathbf{z}' g(\mathbf{z}; \theta) \left(k(\mathbf{z}, \mathbf{z}') - k_t(\mathbf{z})^\top C_t k_t(\mathbf{z}') \right) g(\mathbf{z}'; \theta)^\top,\end{aligned}\tag{11.21}$$

where \hat{Q}_t and \hat{S}_t are the posterior moments of Q computed by the GPTD critic from Eq. 11.9.

These equations provide us with the general form of the posterior policy gradient moments. We are now left with a computational issue, namely, how to compute the following integrals appearing in these expressions?

$$U_t = \int_{\mathcal{Z}} d\mathbf{z} g(\mathbf{z}; \theta) k_t(\mathbf{z})^\top \quad \text{and} \quad V = \int_{\mathcal{Z}^2} d\mathbf{z} d\mathbf{z}' g(\mathbf{z}; \theta) k(\mathbf{z}, \mathbf{z}') g(\mathbf{z}'; \theta)^\top.\tag{11.22}$$

Using the definitions in Eq. 11.22, we may write the gradient posterior moments compactly as

$$\mathbb{E}[\nabla \eta(\theta)|\mathcal{D}_t] = U_t \alpha_t \quad \text{and} \quad \mathbf{Cov}[\nabla \eta(\theta)|\mathcal{D}_t] = V - U_t C_t U_t^\top.\tag{11.23}$$

Ghavamzadeh and Engel (2007) showed that in order to render these integrals analytically tractable, the prior covariance kernel should be defined as $k(\mathbf{z}, \mathbf{z}') = k_s(s, s') + k_F(\mathbf{z}, \mathbf{z}')$, the sum of an arbitrary state-kernel k_s and the Fisher kernel between state-action pairs $k_F(\mathbf{z}, \mathbf{z}') = u(\mathbf{z})^\top G(\theta)^{-1} u(\mathbf{z}')$. They proved that using this prior covariance kernel, U_t and V from Eq. 11.22 satisfy $U_t = [u(\mathbf{z}_0), \dots, u(\mathbf{z}_t)]$ and $V = G(\theta)$. When the posterior moments of the gradient of the expected return are available, a Bayesian actor-critic (BAC) algorithm can be easily derived by updating the policy parameters in the direction of the mean.

Similar to the BPG case in Section 11.2.2, the Fisher information matrix of each policy may be estimated using MC or ML methods, and the algorithm may be made significantly more efficient, both in time and memory, and more numerically stable by sparsifying the solution using for example the online sparsification method of Engel et al (2002).

11.3 Model-Based Bayesian Reinforcement Learning

In model-based RL we explicitly estimate a model of the environment dynamics while interacting with the system. In model-based Bayesian RL we start with a prior belief over the unknown parameters of the MDP model. Then, when a realization of an unknown parameter is observed while interacting with the environment, we update the belief to reflect the observed data. In the case of discrete state-action MDPs, each unknown transition probability $P(s'|s,a)$ is an unknown parameter $\theta_a^{s,s'}$ that takes values in the $[0,1]$ interval; consequently beliefs are probability densities over continuous intervals. Model-based approaches tend to be more complex computationally than model-free ones, but they allow for prior knowledge of the environment to be more naturally incorporated in the learning process.

11.3.1 POMDP Formulation of Bayesian RL

We can formulate model-based Bayesian RL as a partially observable Markov decision process (POMDP) (Duff, 2002), which is formally described by a tuple $\langle \mathcal{S}_P, \mathcal{A}_P, \mathcal{O}_P, T_P, Z_P, R_P \rangle$. Here $\mathcal{S}_P = \mathcal{S} \times \{\theta_a^{s,s'}\}$ is the hybrid set of states defined by the cross product of the (discrete and fully observable) nominal MDP states s and the (continuous and unobserved) model parameters $\theta_a^{s,s'}$ (one parameter for each feasible state-action-state transition of the MDP). The action space of the POMDP $\mathcal{A}_P = \mathcal{A}$ is the same as that of the MDP. The observation space $\mathcal{O}_P = \mathcal{S}$ coincides with the MDP state space since the latter is fully observable. The transition function $T_P(s, \theta, a, s', \theta') = P(s', \theta' | s, \theta, a)$ can be factored in two conditional distributions, one for the MDP states $P(s' | s, \theta_a^{s,s'}, a) = \theta_a^{s,s'}$, and one for the unknown parameters $P(\theta' | \theta) = \delta_\theta(\theta')$ where $\delta_\theta(\theta')$ is a Kronecker delta with value 1 when $\theta' = \theta$ and value 0 otherwise. This Kronecker delta reflects the assumption that unknown parameters are stationary, i.e., θ does not change with time. The observation function $Z_P(s', \theta', a, o) = P(o | s', \theta', a)$ indicates the probability of making an observation o when joint state s', θ' is reached after executing action a . Since the observations are the MDP states, then $P(o | s', \theta', a) = \delta_{s'}(o)$.

We can formulate a *belief-state MDP* over this POMDP by defining beliefs over the unknown parameters $\theta_a^{s,s'}$. The key point is that this belief-state MDP is *fully observable* even though the original RL problem involves hidden quantities. This formulation effectively turns the reinforcement learning problem into a planning problem in the space of beliefs over the unknown MDP model parameters.

For discrete MDPs a natural representation of beliefs is via Dirichlet distributions, as Dirichlets are conjugate densities of multinomials (DeGroot, 1970). A Dirichlet distribution $Dir(p; n) \propto \prod_i p_i^{n_i - 1}$ over a multinomial p is parameterized by positive numbers n_i , such that $n_i - 1$ can be interpreted as the number of times that the p_i -probability event has been observed. Since each feasible transition s, a, s'

pertains only to one of the unknowns, we can model beliefs as products of Dirichlets, one for each unknown model parameter $\theta_a^{s,s'}$.

Belief monitoring in this POMDP corresponds to Bayesian updating of the beliefs based on observed state transitions. For a *prior* belief $b(\theta) = \text{Dir}(\theta; n)$ over some transition parameter θ , when a specific (s, a, s') transition is observed in the environment, the *posterior* belief is analytically computed by the Bayes' rule, $b'(\theta) \propto \theta_a^{s,s'} b(\theta)$. If we represent belief states by a tuple $\langle s, \{n_a^{s,s'}\} \rangle$ consisting of the current state s and the hyperparameters $n_a^{s,s'}$ for each Dirichlet, belief updating simply amounts to setting the current state to s' and incrementing by one the hyperparameter $n_a^{s,s'}$ that matches the observed transition s, a, s' .

The POMDP formulation of Bayesian reinforcement learning provides a natural framework to reason about the exploration/exploitation tradeoff. Since beliefs encode all the information gained by the learner (i.e., sufficient statistics of the history of past actions and observations) and an optimal POMDP policy is a mapping from beliefs to actions that maximizes the expected total rewards, it follows that an optimal POMDP policy naturally optimizes the exploration/exploitation tradeoff. In other words, since the goal in balancing exploitation (immediate gain) and exploration (information gain) is to maximize the overall sum of rewards, then the best tradeoff is achieved by the best POMDP policy. Note however that this assumes that the prior belief is accurate and that computation is exact, which is rarely the case in practice. Nevertheless, the POMDP formulation provides a useful formalism to design algorithms that naturally tradeoff the exploration/exploitation tradeoff.

The POMDP formulation reduces the RL problem to a planning problem with special structure. In the next section we derive the parameterization of the optimal value function, which can be computed exactly by dynamic programming (Poupart et al, 2006). However, since the complexity grows exponentially with the planning horizon, we also discuss some approximations.

11.3.2 Bayesian RL via Dynamic Programming

Using the fact that POMDP observations in Bayesian RL correspond to nominal MDP states, Bellman's equation for the optimal value function in the belief-state MDP reads (Duff, 2002)

$$V_s^*(b) = \max_a R(s, a) + \gamma \sum_{s'} P(s'|s, b, a) V_{s'}^*(b_a^{s,s'}). \quad (11.24)$$

Here s is the current nominal MDP state, b is the current belief over the model parameters θ , and $b_a^{s,s'}$ is the updated belief after transition s, a, s' . The transition model is defined as

$$P(s'|s, b, a) = \int_{\theta} d\theta b(\theta) P(s'|s, \theta, a) = \int_{\theta} d\theta b(\theta) \theta_a^{s,s'}, \quad (11.25)$$

and is just the average transition probability $P(s'|s,a)$ with respect to belief b . Since an optimal POMDP policy achieves by definition the highest attainable expected future reward, it follows that such a policy would automatically optimize the exploration/exploitation tradeoff in the original RL problem.

It is known (see, e.g., chapter 12 in this book) that the optimal finite-horizon value function of a POMDP with discrete states and actions is piecewise linear and convex, and it corresponds to the upper envelope of a set Γ of linear segments called α -vectors: $V^*(b) = \max_{\alpha \in \Gamma} \alpha(b)$. In the literature, α is both defined as a linear function of b (i.e., $\alpha(b)$) and as a vector of s (i.e., $\alpha(s)$) such that $\alpha(b) = \sum_s b(s)\alpha(s)$. Hence, for discrete POMDPs, value functions can be parameterized by a set of α -vectors each represented as a vector of values for each state. Conveniently, this parameterization is closed under Bellman backups.

In the case of Bayesian RL, despite the hybrid nature of the state space, the piecewise linearity and convexity of the value function may still hold as demonstrated by Duff (2002) and Porta et al (2005). In particular, the optimal finite-horizon value function of a discrete-action POMDP corresponds to the upper envelope of a set Γ of linear segments called α -functions (due to the continuous nature of the POMDP state θ), which can be grouped in subsets per nominal state s :

$$V_s^*(b) = \max_{\alpha \in \Gamma} \alpha_s(b). \quad (11.26)$$

Here α can be defined as a linear function of b subscripted by s (i.e., $\alpha_s(b)$) or as a function of θ subscripted by s (i.e., $\alpha_s(\theta)$) such that

$$\alpha_s(b) = \int_{\theta} d\theta b(\theta) \alpha_s(\theta). \quad (11.27)$$

Hence value functions in Bayesian RL can also be parameterized as a set of α -functions. Moreover, similarly to discrete POMDPs, the α -functions can be updated by Dynamic Programming (DP) as we will show next. However, in Bayesian RL the representation of α -functions grows in complexity with the number of DP backups: For horizon T , the optimal value function may involve a number of α -functions that is exponential in T , but also each α -function will have a representation complexity (for instance, number of nonzero coefficients in a basis function expansion) that is also exponential in T , as we will see next.

11.3.2.1 Value Function Parameterization

Suppose that the optimal value function $V_s^k(b)$ for k steps-to-go is composed of a set Γ^k of α -functions such that $V_s^k(b) = \max_{\alpha \in \Gamma^k} \alpha_s(b)$. Using Bellman's equation, we can compute by dynamic programming the best set Γ^{k+1} representing the optimal value function V^{k+1} with $k+1$ stages-to-go. First we rewrite Bellman's equation (Eq. 11.24) by substituting V^k for the maximum over the α -functions in Γ^k as in Eq. 11.26:

$$V_s^{k+1}(b) = \max_a R(b,a) + \gamma \sum_{s'} P(s'|s,b,a) \max_{\alpha \in \Gamma^k} \alpha_{s'}(b_a^{s,s'}).$$

Then we decompose Bellman's equation in three steps. The first step finds the maximal α -function for each a and s' . The second step finds the best action a . The third step performs the actual Bellman backup using the maximal action and α -functions:

$$\alpha_{b,a}^{s,s'} = \arg \max_{\alpha \in \Gamma^k} \alpha(b_a^{s,s'}) \quad (11.28)$$

$$a_b^s = \arg \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,b,a) \alpha_{b,a}^{s,s'}(b_a^{s,s'}) \quad (11.29)$$

$$V_s^{k+1}(b) = R(s,a_b^s) + \gamma \sum_{s'} P(s'|s,b,a_b^s) \alpha_{b,a_b^s}^{s,s'}(b_a^{s,s'}) \quad (11.30)$$

We can further rewrite the third step by using α -functions in terms of θ (instead of b) and expanding the belief state $b_{a_b^s}^{s,s'}$:

$$V_s^{k+1}(b) = R(s,a_b^s) + \gamma \sum_{s'} P(s'|s,b,a_b^s) \int_{\theta} d\theta b_{a_b^s}^{s,s'}(\theta) \alpha_{b,a_b^s}^{s,s'}(\theta) \quad (11.31)$$

$$= R(s,a_b^s) + \gamma \sum_{s'} P(s'|s,b,a_b^s) \int_{\theta} d\theta \frac{b(\theta) P(s'|s,\theta,a_b^s)}{P(s'|s,b,a_b^s)} \alpha_{b,a_b^s}^{s,s'}(\theta) \quad (11.32)$$

$$= R(s,a_b^s) + \gamma \sum_{s'} \int_{\theta} d\theta b(\theta) P(s'|s,\theta,a_b^s) \alpha_{b,a_b^s}^{s,s'}(\theta) \quad (11.33)$$

$$= \int_{\theta} d\theta b(\theta) [R(s,a_b^s) + \gamma \sum_{s'} P(s'|s,\theta,a_b^s) \alpha_{b,a_b^s}^{s,s'}(\theta)] \quad (11.34)$$

The expression in square brackets is a function of s and θ , so we can use it as the definition of an α -function in Γ^{k+1} :

$$\alpha_{b,s}(\theta) = R(s,a_b^s) + \gamma \sum_{s'} P(s'|s,\theta,a_b^s) \alpha_{b,a_b^s}^{s,s'}(\theta). \quad (11.35)$$

For every b we define such an α -function, and together all $\alpha_{b,s}$ form the set Γ^{k+1} . Since each $\alpha_{b,s}$ was defined by using the optimal action and α -functions in Γ^k , it follows that each $\alpha_{b,s}$ is necessarily optimal at b and we can introduce a max over all α -functions with no loss:

$$V_s^{k+1}(b) = \int_{\theta} d\theta b(\theta) \alpha_{b,s}(\theta) = \alpha_s(b) = \max_{\alpha \in \Gamma^{k+1}} \alpha_s(b). \quad (11.36)$$

Based on the above we can show the following (we refer to the original paper for the proof):

Theorem 11.1 (Poupart et al (2006)). *The α -functions in Bayesian RL are linear combinations of products of (unnormalized) Dirichlets.*

Note that in this approach the representation of α -functions grows in complexity with the number of DP backups: Using the above theorem and Eq. 11.35, one can see that the number of components of each α -function grow in each backup by a factor $O(|\mathcal{S}|)$, which yields a number of components that grows exponentially with the planning horizon. In order to mitigate the exponential growth in the number of components, we can project linear combinations of components onto a smaller number of components (e.g., a monomial basis). Poupart et al (2006) describe various projection schemes that achieve that.

11.3.2.2 Exact and Approximate DP Algorithms

Having derived a representation for α -functions that is closed under Bellman backups, one can now transfer several of the algorithms for discrete POMDPs to Bayesian RL. For instance, one can compute an optimal finite-horizon Bayesian RL controller by resorting to a POMDP solution technique akin to Monahan’s enumeration algorithm (see chapter 12 in this book), however in each backup the number of supporting α -functions will in general be an exponential function of $|\mathcal{S}|$.

Alternatively, one can devise approximate (point-based) value iteration algorithms that exploit the value function parameterization via α -functions. For instance, Poupart et al (2006) proposed the BEETLE algorithm for Bayesian RL, which is an extension of the Perseus algorithm for discrete POMDPs (Spaan and Vlassis, 2005). In this algorithm, a set of reachable (s,b) pairs is sampled by simulating several runs of a random policy. Then (approximate) value iteration is done by performing point-based backups at the sampled (s,b) pairs, pertaining to the particular parameterization of the α -functions.

The use of α -functions in value iteration allows for the design of *offline* (i.e., pre-compiled) solvers, as the α -function parameterization offers a generalization to off-sample regions of the belief space. BEETLE is the only known algorithm in the literature that exploits the form of the α -functions to achieve generalization in model-based Bayesian RL. Alternatively, one can use any generic function approximator. For instance, Duff (2003) describes and actor-critic algorithms that approximates the value function with a linear combination of features in (s,θ) . Most other model-based Bayesian RL algorithms are *online* solvers that do not explicitly parameterize the value function. We briefly describe some of these algorithms next.

11.3.3 Approximate Online Algorithms

Online algorithms attempt to approximate the Bayes optimal action by reasoning over the current belief, which often results in myopic action selection strategies. This approach avoids the overhead of offline planning (as with BEETLE), but it may require extensive deliberation at runtime that can be prohibitive in practice.

Early approximate online RL algorithms were based on confidence intervals (Kaelbling, 1993; Meuleau and Bourgine, 1999; Wiering, 1999) or the value of perfect information (VPI) criterion for action selection (Dearden et al, 1999), both resulting in myopic action selection strategies. The latter involves estimating the distribution of optimal Q-values for the MDPs in the support of the current belief, which are then used to compute the expected ‘gain’ for switching from one action to another, hopefully better, action. Instead of building an explicit distribution over Q-values (as in Section 11.2.1.1), we can use the distribution over models $P(\theta)$ to sample models and compute the optimal Q-values of each model. This yields a sample of Q-values that approximates the underlying distribution over Q-values. The exploration gain of each action can then be estimated according to Eq. 11.2, where the expectation over Q-values is approximated by the sample mean. Similar to Eq. 11.1, the value of perfect information can be approximated by:

$$VPI(s,a) \approx \frac{1}{\sum_i w_\theta^i} \sum_i w_\theta^i Gain_{s,a}(q_{s,a}^i) \quad (11.37)$$

where the w_θ^i ’s are the importance weights of the sampled models depending on the proposal distribution used. Dearden et al (1999) describe several efficient procedures to sample the models from some proposal distributions that may be easier to work with than $P(\theta)$.

An alternative myopic Bayesian action selection strategy is *Thompson sampling*, which involves sampling just one MDP from the current belief, solve this MDP to optimality (e.g., by Dynamic Programming), and execute the optimal action at the current state (Thompson, 1933; Strens, 2000), a strategy that reportedly tends to over-explore (Wang et al, 2005).

One may achieve a less myopic action selection strategy by trying to compute a near-optimal policy in the belief-state MDP of the POMDP (see previous section). Since this is just an MDP (albeit continuous and with a special structure), one may use any approximate solver for MDPs. Wang et al (2005); Ross and Pineau (2008) have pursued this idea by applying the sparse sampling algorithm of Kearns et al (1999) on the belief-state MDP. This approach carries out an explicit lookahead to the effective horizon starting from the current belief, backing up rewards through the tree by dynamic programming or linear programming (Castro and Precup, 2007), resulting in a near-Bayes-optimal exploratory action. The search through the tree does not produce a policy that will generalize over the belief space however, and a new tree will have to be generated at each time step which can be expensive in practice. Presumably the sparse sampling approach can be combined with an approach that generalizes over the belief space via an α -function parameterization as in BEETLE, although no algorithm of that type has been reported so far.

11.3.4 Bayesian Multi-Task Reinforcement Learning

Multi-task learning (MTL) is an important learning paradigm and has recently been an area of active research in machine learning (e.g., Caruana 1997; Baxter 2000).

A common setup is that there are multiple related tasks for which we are interested in improving the performance over individual learning by sharing information across the tasks. This transfer of information is particularly important when we are provided with only a limited number of data to learn each task. Exploiting data from related problems provides more training samples for the learner and can improve the performance of the resulting solution. More formally, the main objective in MTL is to maximize the improvement over individual learning averaged over the tasks. This should be distinguished from transfer learning in which the goal is to learn a suitable bias for a class of tasks in order to maximize the expected future performance.

Most RL algorithms often need a large number of samples to solve a problem and cannot directly take advantage of the information coming from other similar tasks. However, recent work has shown that transfer and multi-task learning techniques can be employed in RL to reduce the number of samples needed to achieve nearly-optimal solutions. All approaches to multi-task RL (MTRL) assume that the tasks share similarity in some components of the problem such as dynamics, reward structure, or value function. While some methods explicitly assume that the shared components are drawn from a common generative model (Wilson et al, 2007; Mehta et al, 2008; Lazaric and Ghavamzadeh, 2010), this assumption is more implicit in others (Taylor et al, 2007; Lazaric et al, 2008). In Mehta et al (2008), tasks share the same dynamics and reward features, and only differ in the weights of the reward function. The proposed method initializes the value function for a new task using the previously learned value functions as a prior. Wilson et al (2007) and Lazaric and Ghavamzadeh (2010) both assume that the distribution over some components of the tasks is drawn from a hierarchical Bayesian model (HBM). We describe these two methods in more details below.

Lazaric and Ghavamzadeh (2010) study the MTRL scenario in which the learner is provided with a number of MDPs with common state and action spaces. For any given policy, only a small number of samples can be generated in each MDP, which may not be enough to accurately evaluate the policy. In such a MTRL problem, it is necessary to identify classes of tasks with similar structure and to learn them jointly. It is important to note that here a task is a pair of MDP and policy such that all the MDPs have the same state and action spaces. They consider a particular class of MTRL problems in which the tasks *share structure in their value functions*. To allow the value functions to share a common structure, it is assumed that they are all sampled from a common prior. They adopt the GPTD value function model (see Section 11.2.1) for each task, model the distribution over the value functions using a HBM, and develop solutions to the following problems: (i) joint learning of the value functions (multi-task learning), and (ii) efficient transfer of the information acquired in (i) to facilitate learning the value function of a newly observed task (transfer learning). They first present a HBM for the case in which all the value functions belong to the same class, and derive an EM algorithm to find MAP estimates of the value functions and the model's hyper-parameters. However, if the functions do not belong to the same class, simply learning them together can be detrimental (*negative transfer*). It is therefore important to have models that will generally benefit from related tasks and will not hurt performance when the tasks are unrelated. This

is particularly important in RL as changing the policy at each step of policy iteration (this is true even for fitted value iteration) can change the way tasks are clustered together. This means that even if we start with value functions that all belong to the same class, after one iteration the new value functions may be clustered into several classes. To address this issue, they introduce a Dirichlet process (DP) based HBM for the case that the value functions belong to an undefined number of classes, and derive inference algorithms for both the multi-task and transfer learning scenarios in this model.

The MTRL approach in Wilson et al (2007) also uses a DP-based HBM to model the distribution over a common structure of the tasks. In this work, the tasks *share structure in their dynamics and reward function*. The setting is incremental, i.e., the tasks are observed as a sequence, and there is no restriction on the number of samples generated by each task. The focus is not on joint learning with finite number of samples, it is on using the information gained from the previous tasks to facilitate learning in a new one. In other words, the focus in this work is on transfer and not on multi-task learning.

11.3.5 Incorporating Prior Knowledge

When transfer learning and multi-task learning are not possible, the learner may still want to use domain knowledge to reduce the complexity of the learning task. In non-Bayesian reinforcement learning, domain knowledge is often implicitly encoded in the choice of features used to encode the state space, parametric form of the value function, or the class of policies considered. In Bayesian reinforcement learning, the prior distribution provides an explicit and expressive mechanism to encode domain knowledge. Instead of starting with a non-informative prior (e.g., uniform, Jeffrey's prior), one can reduce the need for data by specifying a prior that biases the learning towards parameters that a domain expert feels are more likely.

For instance, in model-based Bayesian reinforcement learning, Dirichlet distributions over the transition and reward distributions can naturally encode an expert's bias. Recall that the hyperparameters $n_i - 1$ of a Dirichlet can be interpreted as the number of times that the p_i -probability event has been observed. Hence, if the expert has access to prior data where each event occurred $n_i - 1$ times or has reasons to believe that each event would occur $n_i - 1$ times in a fictitious experiment, then a corresponding Dirichlet can be used as an informative prior. Alternatively, if one has some belief or prior data to estimate the mean and variance of some unknown multinomial, then the hyperparameters of the Dirichlet can be set by moment matching.

A drawback of the Dirichlet distribution is that it only allows unimodal priors to be expressed. However, mixtures of Dirichlets can be used to express multimodal distributions. In fact, since Dirichlets are monomials (i.e., $\text{Dir}(\theta) = \prod_i \theta_i^{n_i}$), then mixtures of Dirichlets are polynomials with positive coefficients (i.e., $\sum_j c_j \prod_i \theta_i^{n_{ij}}$). So with a large enough number of mixture components it is possible to approximate

arbitrarily closely any desirable prior over an unknown multinomial distribution. Pavlov and Poupart (2008) explored the use of mixtures of Dirichlets to express joint priors over the model dynamics and the policy. Although mixtures of Dirichlets are quite expressive, in some situation it may be possible to structure the priors according to a generative model. To that effect, Doshi-Velez et al (2010) explored the use of hierarchical priors such as hierarchical Dirichlet processes over the model dynamics and policies represented as stochastic finite state controllers. The multi-task and transfer learning techniques described in the previous section also explore hierarchical priors over the value function (Lazaric and Ghavamzadeh, 2010) and the model dynamics (Wilson et al, 2007).

11.4 Finite Sample Analysis and Complexity Issues

One of the main attractive features of the Bayesian approach to RL is the possibility of obtaining finite sample estimation for the statistics of a given policy in terms of posterior expected value and variance. This idea was first pursued by Mannor et al (2007), who considered the bias and variance of the value function estimate of a single policy. Assuming an exogenous sampling process (i.e., we only get to observe the transitions and rewards, but not to control them), there exists a nominal model (obtained by, say, maximum a-posteriori probability estimate) and a posterior probability distribution over all possible models. Given a policy π and a posterior distribution over model $\theta = \langle T, r \rangle$, we can consider the expected posterior value function as:

$$\mathbb{E}_{\tilde{T}, \tilde{r}} \left[\mathbb{E}_s \left[\sum_{t=1}^{\infty} \gamma^t \tilde{r}(s_t) | \tilde{T} \right] \right], \quad (11.38)$$

where the outer expectation is according to the posterior over the parameters of the MDP model and the inner expectation is with respect to transitions given that the model parameters are fixed. Collecting the infinite sum, we get

$$\mathbb{E}_{\tilde{T}, \tilde{r}} \left[(I - \gamma \tilde{T}_\pi)^{-1} \tilde{r}_\pi \right], \quad (11.39)$$

where \tilde{T}_π and \tilde{r}_π are the transition matrix and reward vector of policy π when model $\langle \tilde{T}, \tilde{r} \rangle$ is the true model. This problem maximizes the expected return over both the trajectories and the model random variables. Because of the nonlinear effect of \tilde{T} on the expected return, Mannor et al (2007) argue that evaluating the objective of this problem for a given policy is already difficult.

Assuming a Dirichlet prior for the transitions and a Gaussian prior for the rewards, one can obtain bias and variance estimates for the value function of a given policy. These estimates are based on first order or second order approximations of Equation (11.39). From a computational perspective, these estimates can be easily computed and the value function can be de-biased. When trying to optimize over the policy space, Mannor et al (2007) show experimentally that the common approach

consisting of using the most likely (or expected) parameters leads to a strong bias in the performance estimate of the resulting policy.

The Bayesian view for a finite sample naturally leads to the question of policy optimization, where an additional maximum over all policies is taken in (11.38). The standard approach in Markov decision processes is to consider the so-called robust approach: assume the parameters of the problem belong to some uncertainty set and find the policy with the best worst-case performance. This can be done efficiently using dynamic programming style algorithms; see Niliim and El Ghaoui (2005); Iyengar (2005). The problem with the robust approach is that it leads to over-conservative solutions. Moreover, the currently available algorithms require the uncertainty in different states to be uncorrelated, meaning that the uncertainty set is effectively taken as the Cartesian product of state-wise uncertainty sets.

One of the benefits of the Bayesian perspective is that it enables using certain risk aware approaches since we have a probability distribution on the available models. For example, it is possible to consider bias-variance tradeoffs in this context, where one would maximize reward subject to variance constraints or give a penalty for excessive variance. Mean-variance optimization in the Bayesian setup seems like a difficult problem, and there are currently no known complexity results about it. Curtailing this problem, Delage and Mannor (2010) present an approximation to a risk-sensitive percentile optimization criterion:

$$\begin{aligned} & \text{maximize}_{y \in \mathbb{R}, \pi \in \Upsilon} && y \\ & \text{s.t. } P_\theta (\mathbb{E}_s (\sum_{t=0}^{\infty} \gamma^t r_t(s_t) | s_0 \sim q, \pi) \geq y) \geq 1 - \varepsilon. \end{aligned} \quad (11.40)$$

For a given policy π , the above chance-constrained problem gives us a $1 - \varepsilon$ guarantee that π will perform better than the computed y . The parameter ε in Equation (11.40) measures the risk of the policy doing worse than y . The performance measure we use is related to risk-sensitive criteria often used in finance such as value-at-risk. The program (11.40) is not as conservative as the robust approach (which is derived by taking $\varepsilon = 0$), but also not as optimistic as taking the nominal parameters. From a computational perspective, Delage and Mannor (2010) show that the optimization problem is NP-hard in general, but is polynomially solvable if the reward posterior is Gaussian and there is no uncertainty in the transitions. Still, second order approximations yield a tractable approximation in the general case, if there is a Gaussian prior to the reward and a Dirichlet prior to the transitions.

The above works address policy optimization and evaluation given an exogenous state sampling procedure. It is of interest to consider the exploration-exploitation problem in reinforcement learning (RL) from the sample complexity perspective as well. While the Bayesian approach to model-based RL offers an elegant solution to this problem, by considering a distribution over possible models and acting to maximize expected reward, the Bayesian solution is intractable for all but the simplest problems; see, however, stochastic tree search approximations in Dimitrakakis (2010). Two recent papers address the issue of complexity in model-based BRL. In the first paper, Kolter and Ng (2009) present a simple algorithm, and prove that with high probability it is able to perform approximately close to the true (intractable)

optimal Bayesian policy after a polynomial (in quantities describing the system) number of time steps. The algorithm and analysis are reminiscent to PAC-MDP (e.g., Brafman and Tennenholz (2002); Strehl et al (2006)) but it explores in a greedier style than PAC-MDP algorithms. In the second paper, Asmuth et al (2009) present an approach that drives exploration by sampling multiple models from the posterior and selecting actions optimistically. The decision when to re-sample the set and how to combine the models is based on optimistic heuristics. The resulting algorithm achieves near optimal reward with high probability with a sample complexity that is low relative to the speed at which the posterior distribution converges during learning. Finally, Fard and Pineau (2010) derive a PAC-Bayesian style bound that allows balancing between the distribution-free PAC and the data-efficient Bayesian paradigms.

11.5 Summary and Discussion

While Bayesian Reinforcement Learning was perhaps the first kind of reinforcement learning considered in the 1960s by the Operations Research community, a recent surge of interest by the Machine Learning community has lead to many advances described in this chapter. Much of this interest comes from the benefits of maintaining explicit distributions over the quantities of interest. In particular, the exploration/exploitation tradeoff can be naturally optimized once a distribution is used to quantify the uncertainty about various parts of the model, value function or gradient. Notions of risk can also be taken into account while optimizing a policy.

In this chapter we provided an overview of the state of the art regarding the use of Bayesian techniques in reinforcement learning for a single agent in fully observable domains. We note that Bayesian techniques have also been used in partially observable domains (Ross et al, 2007, 2008; Poupart and Vlassis, 2008; Doshi-Velez, 2009; Veness et al, 2010) and multi-agent systems (Chalkiadakis and Boutilier, 2003, 2004; Gmytrasiewicz and Doshi, 2005).

References

- Aharony, N., Zehavi, T., Engel, Y.: Learning wireless network association control with Gaussian process temporal difference methods. In: Proceedings of OPNETWORK (2005)
- Asmuth, J., Li, L., Littman, M.L., Nouri, A., Wingate, D.: A Bayesian sampling approach to exploration in reinforcement learning. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2009, pp. 19–26. AUAI Press (2009)
- Bagnell, J., Schneider, J.: Covariant policy search. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (2003)
- Barto, A., Sutton, R., Anderson, C.: Neuron-like elements that can solve difficult learning control problems. IEEE Transaction on Systems, Man and Cybernetics 13, 835–846 (1983)

- Baxter, J.: A model of inductive bias learning. *Journal of Artificial Intelligence Research* 12, 149–198 (2000)
- Baxter, J., Bartlett, P.: Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research* 15, 319–350 (2001)
- Bellman, R.: A problem in sequential design of experiments. *Sankhya* 16, 221–229 (1956)
- Bellman, R.: Adaptive Control Processes: A Guided Tour. Princeton University Press (1961)
- Bellman, R., Kalaba, R.: On adaptive control processes. *Transactions on Automatic Control, IRE* 4(2), 1–9 (1959)
- Bhatnagar, S., Sutton, R., Ghavamzadeh, M., Lee, M.: Incremental natural actor-critic algorithms. In: *Proceedings of Advances in Neural Information Processing Systems*, vol. 20, pp. 105–112. MIT Press (2007)
- Bhatnagar, S., Sutton, R., Ghavamzadeh, M., Lee, M.: Natural actor-critic algorithms. *Automatica* 45(11), 2471–2482 (2009)
- Brafman, R., Tennenholtz, M.: R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *JMLR* 3, 213–231 (2002)
- Caruana, R.: Multitask learning. *Machine Learning* 28(1), 41–75 (1997)
- Castro, P., Precup, D.: Using linear programming for Bayesian exploration in Markov decision processes. In: *Proc. 20th International Joint Conference on Artificial Intelligence* (2007)
- Chalkiadakis, G., Boutilier, C.: Coordination in multi-agent reinforcement learning: A Bayesian approach. In: *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 709–716 (2003)
- Chalkiadakis, G., Boutilier, C.: Bayesian reinforcement learning for coalition formation under uncertainty. In: *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1090–1097 (2004)
- Cozzolino, J., Gonzales-Zubieta, R., Miller, R.L.: Markovian decision processes with uncertain transition probabilities. Tech. Rep. Technical Report No. 11, Research in the Control of Complex Systems. Operations Research Center, Massachusetts Institute of Technology (1965)
- Cozzolino, J.M.: Optimal sequential decision making under uncertainty. Master's thesis, Massachusetts Institute of Technology (1964)
- Dearden, R., Friedman, N., Russell, S.: Bayesian Q-learning. In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 761–768 (1998)
- Dearden, R., Friedman, N., Andre, D.: Model based Bayesian exploration. In: *UAI*, pp. 150–159 (1999)
- DeGroot, M.H.: Optimal Statistical Decisions. McGraw-Hill, New York (1970)
- Delage, E., Mannor, S.: Percentile optimization for Markov decision processes with parameter uncertainty. *Operations Research* 58(1), 203–213 (2010)
- Dimitrakakis, C.: Complexity of stochastic branch and bound methods for belief tree search in bayesian reinforcement learning. In: *ICAART* (1), pp. 259–264 (2010)
- Doshi-Velez, F.: The infinite partially observable Markov decision process. In: *Neural Information Processing Systems* (2009)
- Doshi-Velez, F., Wingate, D., Roy, N., Tenenbaum, J.: Nonparametric Bayesian policy priors for reinforcement learning. In: *NIPS* (2010)
- Duff, M.: Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes. PhD thesis, University of Massachusetts Amherst (2002)
- Duff, M.: Design for an optimal probe. In: *ICML*, pp. 131–138 (2003)
- Engel, Y.: Algorithms and representations for reinforcement learning. PhD thesis, The Hebrew University of Jerusalem, Israel (2005)

- Engel, Y., Mannor, S., Meir, R.: Sparse Online Greedy Support Vector Regression. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) ECML 2002. LNCS (LNAI), vol. 2430, pp. 84–96. Springer, Heidelberg (2002)
- Engel, Y., Mannor, S., Meir, R.: Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In: Proceedings of the Twentieth International Conference on Machine Learning, pp. 154–161 (2003)
- Engel, Y., Mannor, S., Meir, R.: Reinforcement learning with Gaussian processes. In: Proceedings of the Twenty Second International Conference on Machine Learning, pp. 201–208 (2005a)
- Engel, Y., Szabo, P., Volkinshtain, D.: Learning to control an octopus arm with Gaussian process temporal difference methods. In: Proceedings of Advances in Neural Information Processing Systems, vol. 18, pp. 347–354. MIT Press (2005b)
- Fard, M.M., Pineau, J.: PAC-Bayesian model selection for reinforcement learning. In: Lafferty, J., Williams, C.K.I., Shawe-Taylor, J., Zemel, R., Culotta, A. (eds.) Advances in Neural Information Processing Systems, vol. 23, pp. 1624–1632 (2010)
- Ghavamzadeh, M., Engel, Y.: Bayesian policy gradient algorithms. In: Proceedings of Advances in Neural Information Processing Systems, vol. 19, MIT Press (2006)
- Ghavamzadeh, M., Engel, Y.: Bayesian Actor-Critic algorithms. In: Proceedings of the Twenty-Fourth International Conference on Machine Learning (2007)
- Gmytrasiewicz, P., Doshi, P.: A framework for sequential planning in multi-agent settings. Journal of Artificial Intelligence Research (JAIR) 24, 49–79 (2005)
- Greensmith, E., Bartlett, P., Baxter, J.: Variance reduction techniques for gradient estimates in reinforcement learning. Journal of Machine Learning Research 5, 1471–1530 (2004)
- Iyengar, G.N.: Robust dynamic programming. Mathematics of Operations Research 30(2), 257–280 (2005)
- Jaakkola, T., Haussler, D.: Exploiting generative models in discriminative classifiers. In: Proceedings of Advances in Neural Information Processing Systems, vol. 11, MIT Press (1999)
- Kaelbling, L.P.: Learning in Embedded Systems. MIT Press (1993)
- Kakade, S.: A natural policy gradient. In: Proceedings of Advances in Neural Information Processing Systems, vol. 14 (2002)
- Kearns, M., Mansour, Y., Ng, A.: A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In: Proc. IJCAI (1999)
- Kolter, J.Z., Ng, A.Y.: Near-bayesian exploration in polynomial time. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, pp. 513–520. ACM, New York (2009)
- Konda, V., Tsitsiklis, J.: Actor-Critic algorithms. In: Proceedings of Advances in Neural Information Processing Systems, vol. 12, pp. 1008–1014 (2000)
- Lazaric, A., Ghavamzadeh, M.: Bayesian multi-task reinforcement learning. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning, pp. 599–606 (2010)
- Lazaric, A., Restelli, M., Bonarini, A.: Transfer of samples in batch reinforcement learning. In: Proceedings of ICML, vol. 25, pp. 544–551 (2008)
- Mannor, S., Simester, D., Sun, P., Tsitsiklis, J.N.: Bias and variance approximation in value function estimates. Management Science 53(2), 308–322 (2007)
- Marbach, P.: Simulated-based methods for Markov decision processes. PhD thesis, Massachusetts Institute of Technology (1998)
- Martin, J.J.: Bayesian decision problems and Markov chains. John Wiley, New York (1967)

- Mehta, N., Natarajan, S., Tadepalli, P., Fern, A.: Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning* 73(3), 289–312 (2008)
- Meuleau, N., Bourgine, P.: Exploration of multi-state environments: local measures and back-propagation of uncertainty. *Machine Learning* 35, 117–154 (1999)
- Nilim, A., El Ghaoui, L.: Robust control of Markov decision processes with uncertain transition matrices. *Operations Research* 53(5), 780–798 (2005)
- O'Hagan, A.: Monte Carlo is fundamentally unsound. *The Statistician* 36, 247–249 (1987)
- O'Hagan, A.: Bayes-Hermite quadrature. *Journal of Statistical Planning and Inference* 29, 245–260 (1991)
- Pavlov, M., Poupart, P.: Towards global reinforcement learning. In: NIPS Workshop on Model Uncertainty and Risk in Reinforcement Learning (2008)
- Peters, J., Schaal, S.: Reinforcement learning of motor skills with policy gradients. *Neural Networks* 21(4), 682–697 (2008)
- Peters, J., Vijayakumar, S., Schaal, S.: Reinforcement learning for humanoid robotics. In: Proceedings of the Third IEEE-RAS International Conference on Humanoid Robots (2003)
- Peters, J., Vijayakumar, S., Schaal, S.: Natural Actor-Critic. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 280–291. Springer, Heidelberg (2005)
- Porta, J.M., Spaan, M.T., Vlassis, N.: Robot planning in partially observable continuous domains. In: Proc. Robotics: Science and Systems (2005)
- Poupart, P., Vlassis, N.: Model-based Bayesian reinforcement learning in partially observable domains. In: International Symposium on Artificial Intelligence and Mathematics, ISAIM (2008)
- Poupart, P., Vlassis, N., Hoey, J., Regan, K.: An analytic solution to discrete Bayesian reinforcement learning. In: Proc. Int. Conf. on Machine Learning, Pittsburgh, USA (2006)
- Rasmussen, C., Ghahramani, Z.: Bayesian Monte Carlo. In: Proceedings of Advances in Neural Information Processing Systems, vol. 15, pp. 489–496. MIT Press (2003)
- Rasmussen, C., Williams, C.: Gaussian Processes for Machine Learning. MIT Press (2006)
- Reisinger, J., Stone, P., Miikkulainen, R.: Online kernel selection for Bayesian reinforcement learning. In: Proceedings of the Twenty-Fifth Conference on Machine Learning, pp. 816–823 (2008)
- Ross, S., Pineau, J.: Model-based Bayesian reinforcement learning in large structured domains. In: Uncertainty in Artificial Intelligence, UAI (2008)
- Ross, S., Chaib-Draa, B., Pineau, J.: Bayes-adaptive POMDPs. In: Advances in Neural Information Processing Systems, NIPS (2007)
- Ross, S., Chaib-Draa, B., Pineau, J.: Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 2845–2851 (2008)
- Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)
- Silver, E.A.: Markov decision processes with uncertain transition probabilities or rewards. Tech. Rep. Technical Report No. 1, Research in the Control of Complex Systems. Operations Research Center, Massachusetts Institute of Technology (1963)
- Spaan, M.T.J., Vlassis, N.: Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research* 24, 195–220 (2005)
- Strehl, A.L., Li, L., Littman, M.L.: Incremental model-based learners with formal learning-time guarantees. In: UAI (2006)
- Strens, M.: A Bayesian framework for reinforcement learning. In: ICML (2000)

- Sutton, R.: Temporal credit assignment in reinforcement learning. PhD thesis, University of Massachusetts Amherst (1984)
- Sutton, R.: Learning to predict by the methods of temporal differences. *Machine Learning* 3, 9–44 (1988)
- Sutton, R., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: *Proceedings of Advances in Neural Information Processing Systems*, vol. 12, pp. 1057–1063 (2000)
- Taylor, M., Stone, P., Liu, Y.: Transfer learning via inter-task mappings for temporal difference learning. *JMLR* 8, 2125–2167 (2007)
- Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 285–294 (1933)
- Veness, J., Ng, K.S., Hutter, M., Silver, D.: Reinforcement learning via AIXI approximation. In: *AAAI* (2010)
- Wang, T., Lizotte, D., Bowling, M., Schuurmans, D.: Bayesian sparse sampling for on-line reward optimization. In: *ICML* (2005)
- Watkins, C.: Learning from delayed rewards. PhD thesis, Kings College, Cambridge, England (1989)
- Wiering, M.: Explorations in efficient reinforcement learning. PhD thesis, University of Amsterdam (1999)
- Williams, R.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 229–256 (1992)
- Wilson, A., Fern, A., Ray, S., Tadepalli, P.: Multi-task reinforcement learning: A hierarchical Bayesian approach. In: *Proceedings of ICML*, vol. 24, pp. 1015–1022 (2007)

Chapter 12

Partially Observable Markov Decision Processes

Matthijs T.J. Spaan

Abstract. For reinforcement learning in environments in which an agent has access to a reliable state signal, methods based on the Markov decision process (MDP) have had many successes. In many problem domains, however, an agent suffers from limited sensing capabilities that preclude it from recovering a Markovian state signal from its perceptions. Extending the MDP framework, partially observable Markov decision processes (POMDPs) allow for principled decision making under conditions of uncertain sensing. In this chapter we present the POMDP model by focusing on the differences with fully observable MDPs, and we show how optimal policies for POMDPs can be represented. Next, we give a review of model-based techniques for policy computation, followed by an overview of the available model-free methods for POMDPs. We conclude by highlighting recent trends in POMDP reinforcement learning.

12.1 Introduction

The Markov decision process model has proven very successful for learning how to act in stochastic environments. In this chapter, we explore methods for reinforcement learning by relaxing one of the limiting factors of the MDP model, namely the assumption that the agent knows with full certainty the state of the environment. Put otherwise, the agent's sensors allow it to perfectly monitor the state at all times, where the state captures all aspects of the environment relevant for optimal decision making. Clearly, this is a strong assumption that can restrict the applicability of the MDP framework. For instance, when certain state features are hidden from

Matthijs T.J. Spaan

Institute for Systems and Robotics, Instituto Superior Técnico,
Av. Rovisco Pais 1, 1049-001 Lisbon, Portugal (Currently at Delft University of Technology,
Delft, The Netherlands).
e-mail: mtjspa@isr.ist.utl.pt

the agent the state signal will no longer be Markovian, violating a key assumption of most reinforcement-learning techniques (Sutton and Barto, 1998).¹

One example of particular interest arises when applying reinforcement learning to embodied agents. In many robotic applications the robot's on-board sensors do not allow it to unambiguously identify its own location or pose (Thrun et al, 2005). Furthermore, a robot's sensors are often limited to observing its direct surroundings, and might not be adequate to monitor those features of the environment's state beyond its vicinity, so-called hidden state. Another source of uncertainty regarding the true state of the system are imperfections in the robot's sensors. For instance, let us suppose a robot uses a camera to identify the person it is interacting with. The face-recognition algorithm processing the camera images is likely to make mistakes sometimes, and report the wrong identity. Such an imperfect sensor also prevents the robot from knowing the true state of the system: even if the vision algorithm reports person A, it is still possible that person B is interacting with the robot. Although in some domains the issues resulting from imperfect sensing might be ignored, in general they can lead to severe performance deterioration (Singh et al, 1994).

Instead, in this chapter we consider an extension of the (fully observable) MDP setting that also deals with uncertainty resulting from the agent's imperfect sensors. A partially observable Markov decision process (POMDP) allows for optimal decision making in environments which are only partially observable to the agent (Kaelbling et al, 1998), in contrast with the full observability mandated by the MDP model. In general the partial observability stems from two sources: (i) multiple states give the same sensor reading, in case the agent can only sense a limited part of the environment, and (ii) its sensor readings are noisy: observing the same state can result in different sensor readings. The partial observability can lead to "perceptual aliasing": different parts of the environment appear similar to the agent's sensor system, but require different actions. The POMDP captures the partial observability in a probabilistic observation model, which relates possible observations to states.

Classic POMDP examples are the machine maintenance (Smallwood and Sondik, 1973) or structural inspection (Ellis et al, 1995) problems. In these types of problems, the agent has to choose when to inspect a certain machine part or bridge section, to decide whether maintenance is necessary. However, to allow for inspection the machine has to be stopped, or the bridge to be closed, which has a clear economic cost. A POMDP model can properly balance the trade-off between expected deterioration over time and scheduling inspection or maintenance activities. Furthermore, a POMDP can model the scenario that only choosing to inspect provides information regarding the state of the machine or bridge, and that some flaws are not always revealed reliably. More recently, the POMDP model has gained in relevance for robotic applications such as robot navigation (Simmons and Koenig, 1995; Spaan and Vlassis, 2004; Roy et al, 2005; Foka and Trahanias, 2007), active sensing (Hoey and Little, 2007; Spaan et al, 2010), object grasping (Hsiao et al, 2007) or human-robot interaction (Doshi and Roy, 2008). Finally, POMDPs have been

¹ Note to editor: this point is most likely mentioned before in the book, for reasons of coherence this citation can be replaced with a reference to the correct section.

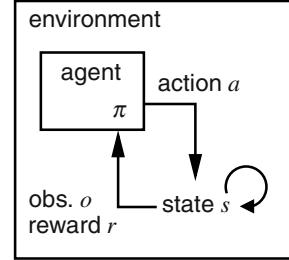


Fig. 12.1 A POMDP agent interacting with its environment

applied in diverse domains such as treatment planning in medicine (Hauskrecht and Fraser, 2000), spoken dialogue systems (Williams and Young, 2007), developing navigation aids (Stankiewicz et al, 2007), or invasive species management (Haight and Polasky, 2010).

The remainder of this chapter is organized as follows. First, in Section 12.2 we formally introduce the POMDP model, and we show that the partial observability leads to a need for memory or internal state on the part of the agent. We discuss how optimal policies and value functions are represented in the POMDP framework. Next, Section 12.3 reviews model-based techniques for POMDPs, considering optimal, approximate and heuristic techniques. Section 12.4 gives an overview of the model-free reinforcement learning techniques that have been developed for or can be applied to POMDPs. Finally, Section 12.5 describes some recent developments in POMDP reinforcement learning.

12.2 Decision Making in Partially Observable Environments

In this section we formally introduce the POMDP model and related decision-making concepts.

12.2.1 POMDP Model

A POMDP shares many elements with the fully observable MDP model as described in Section 1.3, which we will repeat for completeness. Time is discretized in steps, and at the start of each time step the agent has to execute an action. We will consider only discrete, finite, models, which are by far the most commonly used in the POMDP literature given the difficulties involved with solving continuous models. For simplicity, the environment is represented by a finite set of states $S = \{s^1, \dots, s^N\}$. The set of possible actions $A = \{a^1, \dots, a^K\}$ represent the possible ways the agent can influence the system state. Each time step the agent takes an action a in state s , the environment transitions to state s' according to the probabilistic transition function $T(s, a, s')$ and the agent receives an immediate reward $R(s, a, s')$.

What distinguishes a POMDP from a fully observable MDP is that the agent now perceives an observation $o \in \Omega$, instead of observing s' directly. The discrete set of observations $\Omega = \{o^1, \dots, o^M\}$ represent all possible sensor readings the agent can receive. Which observation the agent receives depends on the next state s' and may also be conditional on its action a , and is drawn according to the observation function $O : S \times A \times \Omega \rightarrow [0,1]$. The probability of observing o in state s' after executing a is $O(s', a, o)$. In order for O to be a valid probability distribution over possible observations it is required that $\forall s' \in S, a \in A, o \in \Omega O(s', a, o) \geq 0$ and that $\sum_{o \in \Omega} O(s', a, o) = 1$. Alternatively, the observation function can also be defined as $O : S \times \Omega \rightarrow [0,1]$ reflecting domains in which the observation is independent of the last action.²

As in an MDP, the goal of the agent is to act in such a way as to maximize some form of expected long-term reward, for instance

$$E \left[\sum_{t=0}^h \gamma^t R_t \right], \quad (12.1)$$

where $E[\cdot]$ denotes the expectation operator, h is the planning horizon, and γ is a discount rate, $0 \leq \gamma < 1$.

Analogous to Definition 1.3.1, we can define a POMDP as follows.

Definition 12.2.1. A *partially observable Markov decision process* is a tuple $\langle S, A, \Omega, T, O, R \rangle$ in which S is a finite set of states, A is a finite set of actions, Ω is a finite set of observations, T is a transition function defined as $T : S \times A \times S \rightarrow [0,1]$, O is an observation function defined as $O : S \times A \times \Omega \rightarrow [0,1]$ and R is a reward function defined as $R : S \times A \times S \rightarrow \mathbb{R}$.

Fig. 12.1 illustrates these concepts by depicting a schematic representation of a POMDP agent interacting with the environment.

To illustrate how the observation function models different types of partial observability, consider the following examples, which assume a POMDP with 2 states, 2 observations, and 1 action (omitted for simplicity). The case that sensors make mistakes or are noisy can be modeled as follows. For instance,

$$O(s^1, o^1) = 0.8, \quad O(s^1, o^2) = 0.2, \quad O(s^2, o^1) = 0.2, \quad O(s^2, o^2) = 0.8,$$

models an agent equipped with a sensor that is correct in 80% of the cases. When the agent observes o^1 or o^2 , it does not know for sure that the environment is in state s^1 resp. s^2 . The possibility that the state is completely hidden to the agent can be modeled by assigning the same observation to both states (and observation o^2 is effectively redundant):

$$O(s^1, o^1) = 1.0, \quad O(s^1, o^2) = 0.0, \quad O(s^2, o^1) = 1.0, \quad O(s^2, o^2) = 0.0.$$

² Technically speaking, by including the last action taken as a state feature, observation functions of the form $O(s', o)$ can express the same models compared to $O(s', a, o)$ functions.

When the agent receives observation o^1 it is not able to tell whether the environment is in state s^1 or s^2 , which models the hidden state adequately.

12.2.2 Continuous and Structured Representations

As mentioned before, the algorithms presented in this chapter operate on discrete POMDPs, in which state, action, and observation spaces can be represented by finite sets. Here we briefly discuss work on continuous as well as structured POMDP representations, which can be relevant for several applications.

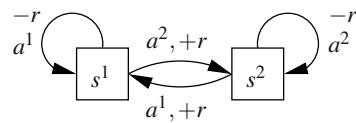
Many real-world POMDPs are more naturally modeled using continuous models (Porta et al, 2006; Brunskill et al, 2008), for instance a robot's pose is often described by continuous (x,y,θ) coordinates. Standard solution methods such as value iteration can also be defined for continuous state spaces (Porta et al, 2005), and continuous observation spaces (Hoey and Poupart, 2005) as well as continuous actions (Spaan and Vlassis, 2005b) have been studied. However, beliefs, observation, action and reward models defined over continuous spaces can have arbitrary forms that may not be parameterizable. In order to design feasible algorithms it is crucial to work with models that have simple parameterizations and result in closed belief updates and Bellman backups. For instance, Gaussian mixtures or particle-based representations can be used for representing beliefs and linear combinations of Gaussians for the models (Porta et al, 2006). As an alternative, simulation-based methods are often capable of dealing with continuous state and action spaces (Thrun, 2000; Ng and Jordan, 2000; Baxter and Bartlett, 2001).

Returning to finite models, in many domains a more structured POMDP representation is beneficial compared to a flat representation (in which all sets are enumerated). Dynamic Bayesian networks are commonly used as a factored POMDP representation (Boutilier and Poole, 1996; Hansen and Feng, 2000), in addition to which algebraic decision diagrams can provide compact model and policy representation (Poupart, 2005; Shani et al, 2008). Relational representations, as described in Chapter 8, have also been proposed for the POMDP model (Sanner and Kersting, 2010; Wang and Kharden, 2010). Furthermore, in certain problems structuring the decision making in several hierarchical levels (see Chapter 9) can allow for improved scalability (Pineau and Thrun, 2002; Theocharous and Mahadevan, 2002; Foka and Trahanias, 2007; Sridharan et al, 2010). Finally, in the case when multiple agents are executing a joint task in a partially observable and stochastic environment, the Decentralized POMDP model can be applied (Bernstein et al, 2002; Seuken and Zilberstein, 2008; Oliehoek et al, 2008), see Chapter 15.

12.2.3 Memory for Optimal Decision Making

As the example in Section 12.2.1 illustrated, in a POMDP the agent's observations do not uniquely identify the state of the environment. However, as the rewards

Fig. 12.2 A two-state POMDP from (Singh et al, 1994), in which the agent receives the same observation in both states.



are still associated with the environment state, as well as the state transitions, a single observation is not a Markovian state signal. In particular, a direct mapping of observations to actions is not sufficient for optimal behavior. In order for an agent to choose its actions successfully in partially observable environments memory is needed.

To illustrate this point, consider the two-state infinite-horizon POMDP depicted in Fig. 12.2 (Singh et al, 1994). The agent has two actions, one of which will deterministically transport it to the other state, while executing the other action has no effect on the state. If the agent jumps to the other state it receives a reward of $r > 0$, and $-r$ otherwise. The optimal policy in the underlying MDP has a value of $\frac{r}{1-\gamma}$, as the agent can gather a reward of r at each time step. In the POMDP however, the agent receives the same observation in both states. As a result, there are only two memoryless deterministic stationary policies possible: always execute a^1 or always execute a^2 . The maximum expected reward of these policies is $r - \frac{\gamma r}{1-\gamma}$, when the agent successfully jumps to the other state at the first time step. If we allow stochastic policies, the best stationary policy would yield an expected discounted reward of 0, when it chooses either action 50% of the time. However, if the agent could remember what actions it had executed, it could execute a policy that alternates between executing a^1 and a^2 . Such a memory-based policy would gather $\frac{\gamma r}{1-\gamma} - r$ in the worst case, which is close to the optimal value in the MDP (Singh et al, 1994).

This example illustrates the need for memory when considering optimal decision making in a POMDP. A straightforward implementation of memory would be to simply store the sequence of actions executed and observations received. However, such a form of memory can grow indefinitely over time, turning it impractical for long planning horizons. Fortunately, a better option exists, as we can transform the POMDP to a belief-state MDP in which the agent summarizes all information about its past using a belief vector $b(s)$ (Stratonovich, 1960; Dynkin, 1965; rAström, 1965). This transformation requires that the transition and observation functions are known to the agent, and hence can be applied only in model-based RL methods.

The belief b is a probability distribution over S , which forms a Markovian signal for the planning task. Given an appropriate state space, the belief is a sufficient statistic of the history, which means the agent could not do any better even if it had remembered the full history of actions and observations. All beliefs are contained in a $(|S| - 1)$ -dimensional simplex $\Delta(S)$, hence we can represent a belief using $|S| - 1$ numbers. Each POMDP problem assumes an initial belief b^0 , which for instance can be set to a uniform distribution over all states (representing complete ignorance regarding the initial state of the environment). Every time the agent takes an action a and observes o , its belief is updated by Bayes' rule:

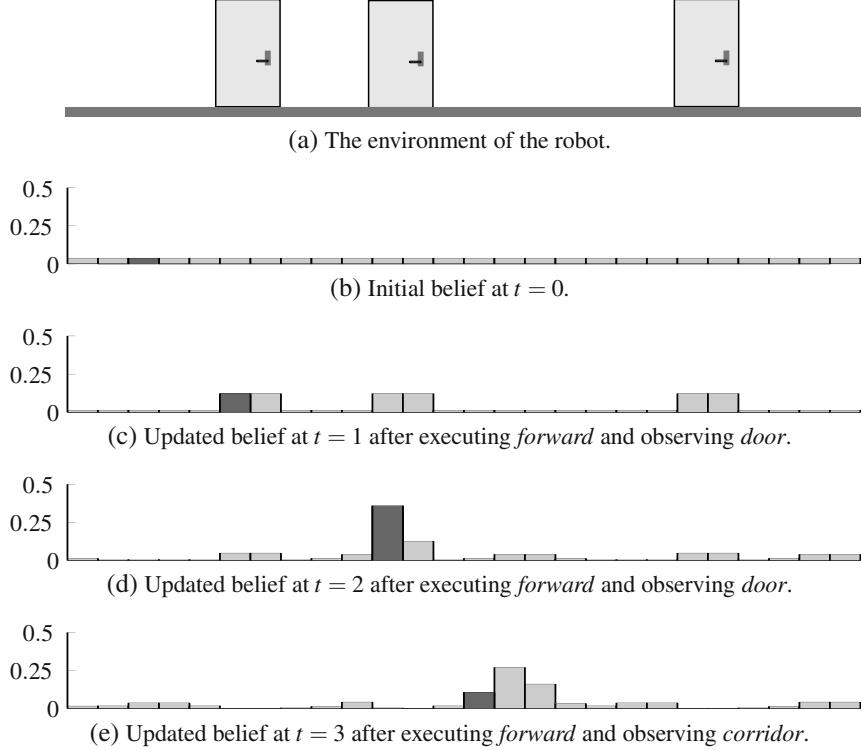


Fig. 12.3 Belief-update example (adapted from Fox et al (1999)). (a) A robot moves in a one-dimensional corridor with three identical doors. (b)-(e) The evolution of the belief over time, for details see main text.

$$b^{ao}(s') = \frac{p(o|s',a)}{p(o|b,a)} \sum_{s \in S} p(s'|s,a)b(s), \quad (12.2)$$

where $p(s'|s,a)$ and $p(o|s',a)$ are defined by model parameters T resp. O , and

$$p(o|b,a) = \sum_{s' \in S} p(o|s',a) \sum_{s \in S} p(s'|s,a)b(s) \quad (12.3)$$

is a normalizing constant.

Fig. 12.3 shows an example of a sequence of belief updates for a robot navigating in a corridor with three identical doors. The corridor is discretized in 26 states and is circular, i.e., the right end of the corridor is connected to the left end. The robot can observe either *door* or *corridor*, but its sensors are noisy. When the robot is positioned in front of a door, it observes *door* with probability 0.9 (and *corridor* with probability 0.1). When the robot is not located in front of a door the probability of observing *corridor* is 0.9. The robot has two actions, *forward* and *backward* (right

resp. left in the figure), which transport the robot 3 (20%), 4 (60%), or 5 (20%) states in the corresponding direction. The initial belief b^0 is uniform, as displayed in Fig. 12.3b. Fig. 12.3c through (e) show how the belief of the robot is updated as it executes the *forward* action each time. The true location of the robot is indicated by the dark-gray component of its belief. In Fig. 12.3c we see that the robot is located in front of the first door, and although it is fairly certain it is located in front of a door, it cannot tell which one. However, after taking another move forward it again observes *door*, and now can pinpoint its location more accurately, because of the particular configuration of the three doors (Fig. 12.3d). However, in Fig. 12.3e the belief blurs again, which is due to the noisy transition model and the fact that the *corridor* observation is not very informative in this case.

12.2.4 Policies and Value Functions

As in the fully observable MDP setting, the goal of the agent is to choose actions which fulfill its task as well as possible, i.e., to learn an optimal policy. In POMDPs, an optimal policy $\pi^*(b)$ maps beliefs to actions. Note that, contrary to MDPs, the policy $\pi(b)$ is a function over a continuous set of probability distributions over S . A policy π can be characterized by a value function $V^\pi : \Delta(S) \rightarrow \mathbb{R}$ which is defined as the expected future discounted reward $V^\pi(b)$ the agent can gather by following π starting from belief b :

$$V^\pi(b) = E_\pi \left[\sum_{t=0}^h \gamma^t R(b_t, \pi(b_t)) \middle| b_0 = b \right], \quad (12.4)$$

where $R(b_t, \pi(b_t)) = \sum_{s \in S} R(s, \pi(b_t)) b_t(s)$.

A policy π which maximizes V^π is called an optimal policy π^* ; it specifies for each b the optimal action to execute at the current step, assuming the agent will also act optimally at future time steps. The value of an optimal policy π^* is defined by the optimal value function V^* . It satisfies the Bellman optimality equation

$$V^* = H_{\text{POMDP}} V^*, \quad (12.5)$$

where H_{POMDP} is the Bellman backup operator for POMDPs, defined as:

$$V^*(b) = \max_{a \in A} \left[\sum_{s \in S} R(s, a) b(s) + \gamma \sum_{o \in O} p(o|b, a) V^*(b^{ao}) \right], \quad (12.6)$$

with b^{ao} given by (12.2), and $p(o|b, a)$ as defined in (12.3). When (12.6) holds for every $b \in \Delta(S)$ we are ensured the solution is optimal.

Computing value functions over a continuous belief space might seem intractable at first, but fortunately the value function has a particular structure that we can exploit (Sondik, 1971). It can be parameterized by a finite number of vectors and has a convex shape. The convexity implies that the value of a belief close to one of the

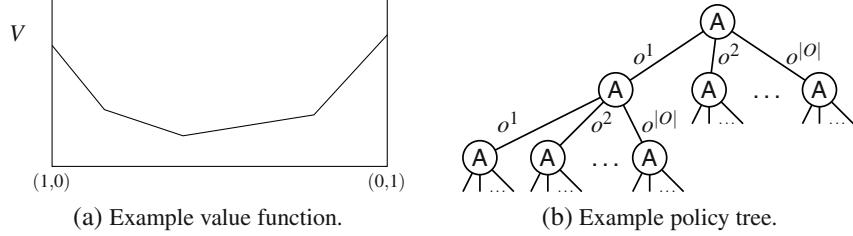


Fig. 12.4 (a) An example of a value function in a two-state POMDP. The y -axis shows the value of each belief, and the x -axis depicts the belief space $\Delta(S)$, ranging from $(1,0)$ to $(0,1)$. (b) An example policy tree, where at a node the agent takes an action, and it transitions to a next node based on the received observation $o \in \{o^1, o^2, \dots, o^{|\mathcal{O}|}\}$.

corners of the belief simplex $\Delta(S)$ will be high. In general, the less uncertainty the agent has over its true state, the better it can predict the future, and as such take better decisions. A belief located exactly at a particular corner of $\Delta(S)$, i.e., $b(s) = 1$ for a particular s , defines with full certainty the state of the agent. In this way, the convex shape of V can be intuitively explained. An example of a convex value function for a two-state POMDP is shown in Fig. 12.4a. As the belief space is a simplex, we can represent any belief in a two-state POMDP on a line, as $b(s^2) = 1 - b(s^1)$. The corners of the belief simplex are denoted by $(1,0)$ and $(0,1)$, which have a higher (or equal) value than a belief in the center of the belief space, e.g., $(0.5,0.5)$.

An alternative way to represent policies in POMDPs is by considering *policy trees* (Kaelbling et al, 1998). Fig. 12.4b shows a partial policy tree, in which the agent starts at the root node of tree. Each node specifies an action which the agent executes at the particular node. Next it receives an observation o , which determines to what next node the agent transitions. The depth of the tree depends on the planning horizon h , i.e., if we want the agent to consider taking h steps, the corresponding policy tree has depth h .

12.3 Model-Based Techniques

If a model of the environment is available, it can be used to compute a policy for the agent. In this section we will discuss several ways of computing POMDP policies, ranging from optimal to approximate and heuristic approaches. Even when the full model is known to the agent, solving the POMDP optimally is typically only computationally feasible for small problems, hence the interest in methods that compromise optimality for reasons of efficiency. All the methods presented in this section exploit a belief state representation (Section 12.2.3), as it provides a compact representation of the complete history of the process.

12.3.1 Heuristics Based on MDP Solutions

First, we discuss some heuristic control strategies that have been proposed which rely on a solution $\pi_{\text{MDP}}^*(s)$ or $Q_{\text{MDP}}^*(s,a)$ of the underlying MDP (Cassandra et al, 1996). The idea is that solving the MDP is of much lower complexity than solving the POMDP (P-complete vs. PSPACE-complete) (Papadimitriou and Tsitsiklis, 1987), but by tracking the belief state still some notion of imperfect state perception can be maintained. Cassandra (1998) provides an extensive experimental comparison of MDP-based heuristics.

Perhaps the most straightforward heuristic is to consider for a belief at a given time step its most likely state (MLS), and use the action the MDP policy prescribes for the state

$$\pi_{\text{MLS}}(b) = \pi_{\text{MDP}}^*(\arg \max_s b(s)). \quad (12.7)$$

The MLS heuristic completely ignores the uncertainty in the current belief, which clearly can be suboptimal.

A more sophisticated approximation technique is Q_{MDP} (Littman et al, 1995), which also treats the POMDP as if it were fully observable. Q_{MDP} solves the MDP and defines a control policy

$$\pi_{Q_{\text{MDP}}}(b) = \arg \max_a \sum_s b(s) Q_{\text{MDP}}^*(s,a). \quad (12.8)$$

Q_{MDP} can be very effective in some domains, but the policies it computes will not take informative actions, as the Q_{MDP} solution assumes that any uncertainty regarding the state will disappear after taking one action. As such, Q_{MDP} policies will fail in domains where repeated information gathering is necessary.

For instance, consider the toy domain in Figure 12.5, which illustrates how MDP-based heuristics can fail (Parr and Russell, 1995). The agent starts in the state marked I , and upon taking any action the system transitions with equal probability to one of two states. In both states it would receive observation A , meaning the agent cannot distinguish between them. The optimal POMDP policy is to take the action a twice in succession, after which the agent is back in the same state. However, because it observed either C or D , it knows in which of the two states marked A it currently is. This knowledge is important for choosing the optimal action (b or c) to transition to the state with positive reward, labelled +1. The fact that the a actions do not change the system state, but only the agent's belief state (two time steps later) is very hard for the MDP-based methods to plan for. It forms an example of reasoning about explicit information gathering effects of actions, for which methods based on MDP solutions do not suffice.

One can also expand the MDP setting to model some form of sensing uncertainty without considering full-blown POMDP beliefs. For instance, in robotics the navigation under localization uncertainty problem can be modeled by the mean and entropy of the belief distribution (Cassandra et al, 1996; Roy and Thrun, 2000).

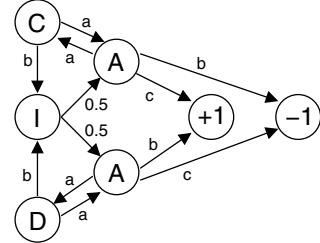


Fig. 12.5 A simple domain in which MDP-based control strategies fail (Parr and Russell, 1995)

Although attractive from a computational perspective, such approaches are likely to fail when the belief is not uni-modal but has a more complex shape.

12.3.2 Value Iteration for POMDPs

To overcome the limitations of MDP-based heuristic methods, we now consider computing optimal POMDP policies via value iteration. The use of belief states allows one to transform the original discrete-state POMDP into a continuous-state MDP. Recall that we can represent a plan in an MDP by its value function, which for every state estimates the amount of discounted cumulative reward the agent can gather when it acts according to the particular plan. In a POMDP the optimal value function, i.e., the value function corresponding to an optimal plan, exhibits particular structure (it is piecewise linear and convex) that one can exploit in order to facilitate computing the solution. Value iteration, for instance, is a method for solving POMDPs that builds a sequence of value-function estimates which converge to the optimal value function for the current task (Sondik, 1971). A value function in a finite-horizon POMDP is parameterized by a finite number of hyperplanes, or vectors, over the belief space, which partition the belief space into a finite amount of regions. Each vector maximizes the value function in a certain region and has an action associated with it, which is the optimal action to take for beliefs in its region.

As we explain next, computing the next value-function estimate—looking one step deeper into the future—requires taking into account all possible actions the agent can take and all subsequent observations it may receive. Unfortunately, this leads to an exponential growth of vectors as the planning horizon increases. Many of the computed vectors will be useless in the sense that their maximizing region is empty, but identifying and subsequently pruning them is an expensive operation.

Exact value-iteration algorithms (Sondik, 1971; Cheng, 1988; Cassandra et al, 1994) search in each value-iteration step the complete belief simplex for a minimal set of belief points that generate the necessary set of vectors for the next-horizon value function. This typically requires linear programming and is therefore costly in high dimensions. Other exact value-iteration algorithms focus on generating all possible next-horizon vectors followed by or interleaved with pruning dominated

vectors in a smart way (Monahan, 1982; Zhang and Liu, 1996; Littman, 1996; Cassandra et al, 1997; Feng and Zilberstein, 2004; Lin et al, 2004; Varakantham et al, 2005). However, pruning again requires linear programming.

The value of an optimal policy π^* is defined by the optimal value function V^* which we compute by iterating a number of stages, at each stage considering a step further into the future. At each stage we apply the exact dynamic-programming operator H_{POMDP} (12.6). If the agent has only one time step left to act, we only have to consider the immediate reward for the particular belief b , and can ignore any future value $V^*(b^{ao})$ and (12.6) reduces to:

$$V_0^*(b) = \max_a \left[\sum_s R(s,a)b(s) \right]. \quad (12.9)$$

We can view the immediate reward function $R(s,a)$ as a set of $|A|$ vectors $\alpha_0^a = (\alpha_0^a(1), \dots, \alpha_0^a(|S|))$, one for each action a : $\alpha_0^a(s) = R(s,a)$. Now we can rewrite (12.9) as follows, where we view b as a $|S|$ -dimensional vector:

$$V_0^*(b) = \max_a \sum_s \alpha_0^a(s)b(s), \quad (12.10)$$

$$= \max_{\{\alpha_0^a\}_a} b \cdot \alpha_0^a, \quad (12.11)$$

where (\cdot) denotes inner product.

In the general case, for $h > 0$, we parameterize a value function V_n at stage n by a finite set of vectors or hyperplanes $\{\alpha_n^k\}$, $k = 1, \dots, |V_n|$. Given a set of vectors $\{\alpha_n^k\}_{k=1}^{|V_n|}$ at stage n , the value of a belief b is given by

$$V_n(b) = \max_{\{\alpha_n^k\}_k} b \cdot \alpha_n^k. \quad (12.12)$$

Additionally, an action $a(\alpha_n^k) \in A$ is associated with each vector, which is the optimal one to take in the current step, for those beliefs for which α_n^k is the maximizing vector. Each vector defines a region in the belief space for which this vector is the maximizing element of V_n . These regions form a partition of the belief space, induced by the piecewise linearity of the value function, as illustrated by Fig. 12.6.

The gradient of the value function at b is given by the vector

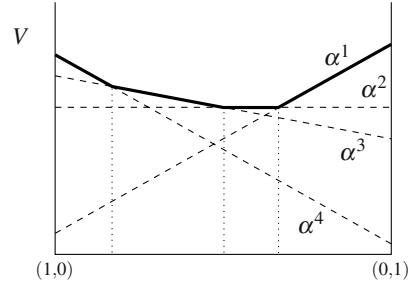
$$\alpha_n^b = \arg \max_{\{\alpha_n^k\}_k} b \cdot \alpha_n^k, \quad (12.13)$$

and the policy at b is given by

$$\pi(b) = a(\alpha_n^b). \quad (12.14)$$

The main idea behind many value-iteration algorithms for POMDPs is that for a given value function V_n and a particular belief point b we can easily compute the vector α_{n+1}^b of $H_{\text{POMDP}}V_n$ such that

Fig. 12.6 Detailed example of a POMDP value function, c.f. Fig. 12.4a. The value function is indicated by the solid black line, and in this case consists of four α vectors, indicated by dashed lines. The induced partitioning of the belief space into four regions is indicated by the vertical dotted lines.



$$\alpha_{n+1}^b = \arg \max_{\{\alpha_{n+1}^k\}_k} b \cdot \alpha_{n+1}^k, \quad (12.15)$$

where $\{\alpha_{n+1}^k\}_{k=1}^{|H_{\text{POMDP}}V_n|}$ is the (unknown) set of vectors for $H_{\text{POMDP}}V_n$. We will denote this operation $\alpha_{n+1}^b = \text{backup}(b)$. For this, we define g_{ao} vectors

$$g_{ao}^k(s) = \sum_{s'} p(o|s', a) p(s'|s, a) \alpha_n^k(s'), \quad (12.16)$$

which represent the vectors resulting from back-projecting α_n^k for a particular a and o . Starting from (12.6) we can derive

$$V_{n+1}(b) = \max_a \left[b \cdot \alpha_0^a + \gamma b \cdot \sum_o \arg \max_{\{g_{ao}^k\}_k} b \cdot g_{ao}^k \right] \quad (12.17)$$

$$= \max_{\{g_a^b\}_a} b \cdot g_a^b, \quad (12.18)$$

$$\text{with } g_a^b = \alpha_0^a + \gamma \sum_o \arg \max_{\{g_{ao}^k\}_k} b \cdot g_{ao}^k, \quad (12.19)$$

which can be re-written as

$$V_{n+1}(b) = b \cdot \arg \max_{\{g_a^b\}_a} b \cdot g_a^b. \quad (12.20)$$

From (12.20) we can derive the vector $\text{backup}(b)$, as this is the vector whose inner product with b yields $V_{n+1}(b)$:

$$\text{backup}(b) = \arg \max_{\{g_a^b\}_{a \in A}} b \cdot g_a^b, \quad (12.21)$$

with g_a^b defined in (12.19). Note that in general not only the computed α vector is retained, but also which action a was the maximizer in (12.21), as that is the optimal action associated with $\text{backup}(b)$.

12.3.3 Exact Value Iteration

The Bellman backup operator (12.21) computes a next-horizon vector for a single belief, and now we will employ this backup operator to compute a complete value function for the next horizon, i.e., one that is optimal for all beliefs in the belief space. Although computing the vector $\text{backup}(b)$ for a given b is straightforward, locating the (minimal) set of points b required to compute *all* vectors $\cup_b \text{backup}(b)$ of $H_{\text{POMDP}}V_n$ is very costly. As each b has a region in the belief space in which its α_n^b is maximal, a family of algorithms tries to identify these regions (Sondik, 1971; Cheng, 1988; Kaelbling et al, 1998). The corresponding b of each region is called a “witness” point, as it testifies to the existence of its region. Other exact POMDP value-iteration algorithms do not focus on searching in the belief space. Instead, they consider enumerating all possible vectors of $H_{\text{POMDP}}V_n$, followed by pruning useless vectors (Monahan, 1982; Zhang and Liu, 1996; Littman, 1996; Cassandra et al, 1997; Feng and Zilberstein, 2004; Lin et al, 2004; Varakantham et al, 2005). We will focus on the enumeration algorithms as they have seen more recent developments and are more commonly used.

12.3.3.1 Monahan’s Enumeration Algorithm

First, we consider the most straightforward way of computing $H_{\text{POMDP}}V_n$, due to Monahan (1982). It involves calculating all possible ways $H_{\text{POMDP}}V_n$ could be constructed, exploiting the known structure of the value function. Note that in each $H_{\text{POMDP}}V_n$ a finite number of vectors are generated, as we have assumed finite sets A and O . We operate independently of a particular b now so (12.19) and hence (12.21) can no longer be applied. Instead of maximizing for all $o \in O$ over the g_{ao}^k vectors for the particular b , we now have to include all ways of selecting g_{ao}^k for all o :

$$H_{\text{POMDP}}V_n = \bigcup_a G_a, \quad \text{with } G_a = \bigoplus_o G_a^o, \quad \text{and } G_a^o = \left\{ \frac{1}{|O|} \alpha_0^a + \gamma g_{ao}^k \right\}_k, \quad (12.22)$$

where \bigoplus denotes the cross-sum operator.³

Unfortunately, at each stage a finite but exponential number of vectors are generated: $|A||V_n|^{|O|}$. The regions of many of the generated vectors will be empty and these vectors are useless as they will not influence the agent’s policy. Technically, they are not part of the value function, and keeping them has no effect on subsequent value functions, apart from the computational burden. Therefore, all value-iteration methods in the enumeration family employ some form of pruning. In particular, Monahan (1982) prunes $H_{\text{POMDP}}V_n$ after computing it:

$$V_{n+1} = \text{prune}(H_{\text{POMDP}}V_n), \quad (12.23)$$

³ Cross sum of sets is defined as: $\bigoplus_k R_k = R_1 \oplus R_2 \oplus \dots \oplus R_k$, with $P \oplus Q = \{ p + q \mid p \in P, q \in Q \}$.

with $H_{\text{POMDP}}V_n$ as defined in (12.22). The `prune` operator is implemented by solving a linear program (White, 1991).

12.3.3.2 Incremental Pruning

Monahan (1982)'s algorithm first generates all $|A||V_n|^{|O|}$ vectors of $H_{\text{POMDP}}V_n$ before pruning all dominated vectors. Incremental Pruning methods (Zhang and Liu, 1996; Cassandra et al, 1997; Feng and Zilberstein, 2004; Lin et al, 2004; Varakantham et al, 2005) save computation time by exploiting the fact that

$$\text{prune}(G \oplus G' \oplus G'') = \text{prune}(\text{prune}(G \oplus G') \oplus G''). \quad (12.24)$$

In this way the number of constraints in the linear program used for pruning grows slowly (Cassandra et al, 1997), leading to better performance. The basic Incremental Pruning algorithm exploits (12.24) when computing V_{n+1} as follows:

$$V_{n+1} = \text{prune}\left(\bigcup_a G_a\right), \quad \text{with} \quad (12.25)$$

$$G_a = \text{prune}\left(\bigoplus_o G_a^o\right) \quad (12.26)$$

$$= \text{prune}(G_a^1 \oplus G_a^2 \oplus G_a^3 \oplus \cdots \oplus G_a^{|O|}) \quad (12.27)$$

$$= \text{prune}(\cdots \text{prune}(\text{prune}(G_a^1 \oplus G_a^2) \oplus G_a^3) \cdots \oplus G_a^{|O|}). \quad (12.28)$$

In general, however, computing exact solutions for POMDPs is an intractable problem (Papadimitriou and Tsitsiklis, 1987; Madani et al, 2003), calling for approximate solution techniques (Lovejoy, 1991; Hauskrecht, 2000). Next we present a family of popular approximate value iteration algorithms.

12.3.4 Point-Based Value Iteration Methods

Given the high computational complexity of optimal POMDP solutions, many methods for approximate solutions have been developed. One powerful idea has been to compute solutions only for those parts of the belief simplex that are reachable, i.e., that can be actually encountered by interacting with the environment. This has motivated the use of approximate solution techniques which focus on the use of a sampled set of *belief points* on which planning is performed (Hauskrecht, 2000; Poon, 2001; Roy and Gordon, 2003; Pineau et al, 2003; Smith and Simmons, 2004; Spaan and Vlassis, 2005a; Shani et al, 2007; Kurniawati et al, 2008), a possibility already mentioned by Lovejoy (1991). The idea is that instead of planning over the complete belief space of the agent (which is intractable for large state spaces), planning is carried out only on a limited set of prototype beliefs B that have been sampled by letting the agent interact with the environment.

As we described before, a major cause of intractability of exact POMDP solution methods is their aim of computing the optimal action for every possible belief point in the belief space $\Delta(S)$. For instance, if we use Monahan's algorithm (12.22) we can end up with a series of value functions whose size grows exponentially in the planning horizon. A natural way to sidestep this intractability is to settle for computing an approximate solution by considering only a finite set of belief points. The backup stage reduces to applying (12.21) a fixed number of times, resulting in a small number of vectors (bounded by the size of the belief set). The motivation for using approximate methods is their ability to compute successful policies for much larger problems, which compensates for the loss of optimality.

The general assumption underlying these so-called *point-based* methods is that by updating not only the value but also its gradient (the α vector) at each $b \in B$, the resulting policy will generalize well and be effective for beliefs outside the set B . Whether or not this assumption is realistic depends on the POMDP's structure and the contents of B , but the intuition is that in many problems the set of 'reachable' beliefs (reachable by following an arbitrary policy starting from the initial belief) forms a low-dimensional manifold in the belief simplex, and thus can be covered densely enough by a relatively small number of belief points.

The basic point-based POMDP update operates as follows. It uses an approximate backup operator \tilde{H}_{PBVI} instead of H_{POMDP} , that in each value-backup stage computes the set

$$\tilde{H}_{\text{PBVI}}V_n = \bigcup_{b \in B} \text{backup}(b), \quad (12.29)$$

using a fixed set of belief points B . An alternative randomized backup operator $\tilde{H}_{\text{PERSEUS}}$ is provided by PERSEUS (Spaan and Vlassis, 2005a), which increases (or at least does not decrease) the value of all belief points in B . The key idea is that in each value-backup stage the value of all points in the belief set B can be improved by only backing up a (randomly selected) subset \tilde{B} of the points:

$$\tilde{H}_{\text{PERSEUS}}V_n = \bigcup_{b \in \tilde{B}} \text{backup}(b), \quad (12.30)$$

$$\text{ensuring that } V_n(b') \leq V_{n+1}(b'), \forall b' \in B. \quad (12.31)$$

In each backup stage the set \tilde{B} is constructed by sampling beliefs from B until the resulting V_{n+1} upper bounds V_n over B , i.e., until condition (12.31) has been met. The $\tilde{H}_{\text{PERSEUS}}$ operator results in value functions with a relatively small number of vectors, allowing for the use of much larger B , which has a positive effect on the approximation accuracy (Pineau et al, 2003).

Crucial to the control quality of the computed approximate solution is the makeup of B . A number of schemes to build B have been proposed. For instance, one could use a regular grid on the belief simplex, computed, e.g., by Freudenthal triangulation (Lovejoy, 1991). Other options include taking all extreme points of the belief simplex or use a random grid (Hauskrecht, 2000; Poon, 2001). An alternative scheme is to include belief points that can be encountered by simulating the

POMDP: we can generate trajectories through the belief space by sampling random actions and observations at each time step (Lovejoy, 1991; Hauskrecht, 2000; Poon, 2001; Pineau et al, 2003; Spaan and Vlassis, 2005a). This sampling scheme focuses the contents of B to be beliefs that can actually be encountered while experiencing the POMDP model.

More intricate schemes for belief sampling have also been proposed. For instance, one can use the MDP solution to guide the belief sampling process (Shani et al, 2007), but in problem domains which require series of information-gathering actions such a heuristic will suffer from similar issues as when using Q_{MDP} (Section 12.3.1). Furthermore, the belief set B does not need to be static, and can be updated while running a point-based solver. HSVI heuristically selects belief points in the search tree starting from the initial belief, based on upper and lower bounds on the optimal value function (Smith and Simmons, 2004, 2005). SARSOP takes this idea a step further by successively approximating the optimal reachable belief space, i.e., the belief space that can be reached by following an optimal policy (Kurniawati et al, 2008).

In general, point-based methods compute solutions in the form of piecewise linear and convex value functions, and given a particular belief, the agent can simply look up which action to take using (12.14).

12.3.5 *Other Approximate Methods*

Besides the point-based methods, other types of approximation structure have been explored as well.

12.3.5.1 *Grid-Based Approximations*

One way to sidestep the intractability of exact POMDP value iteration is to grid the belief simplex, using either a fixed grid (Drake, 1962; Lovejoy, 1991; Bonet, 2002) or a variable grid (Brafman, 1997; Zhou and Hansen, 2001). Value backups are performed for every grid point, but only the value of each grid point is preserved and the gradient is ignored. The value of non-grid points is defined by an interpolation rule. The grid based methods differ mainly on how the grid points are selected and what shape the interpolation function takes. In general, regular grids do not scale well in problems with high dimensionality and non-regular grids suffer from expensive interpolation routines.

12.3.5.2 *Policy Search*

An alternative to computing an (approximate) value function is policy search: these methods search for a good policy within a restricted class of controllers (Platzman, 1981). For instance, policy iteration (Hansen, 1998b) and bounded policy iteration

(BPI) (Poupart and Boutilier, 2004) search through the space of (bounded-size) stochastic finite-state controllers by performing policy-iteration steps. Other options for searching the policy space include gradient ascent (Meuleau et al, 1999a; Kearns et al, 2000; Ng and Jordan, 2000; Baxter and Bartlett, 2001; Aberdeen and Baxter, 2002) and heuristic methods like stochastic local search (Brazunas and Boutilier, 2004). In particular, the PEGASUS method (Ng and Jordan, 2000) estimates the value of a policy by simulating a (bounded) number of trajectories from the POMDP using a fixed random seed, and then takes steps in the policy space in order to maximize this value. Policy search methods have demonstrated success in several cases, but searching in the policy space can often be difficult and prone to local optima (Baxter et al, 2001).

12.3.5.3 Heuristic Search

Another approach for solving POMDPs is based on heuristic search (Satia and Lave, 1973; Hansen, 1998a; Smith and Simmons, 2004). Defining an initial belief b_0 as the root node, these methods build a tree that branches over (a,o) pairs, each of which recursively induces a new belief node. Branch-and-bound techniques are used to maintain upper and lower bounds to the expected return at fringe nodes in the search tree. Hansen (1998a) proposes a policy-iteration method that represents a policy as a finite-state controller, and which uses the belief tree to focus the search on areas of the belief space where the controller can most likely be improved. However, its applicability to large problems is limited by its use of full dynamic-programming updates. As mentioned before, HSVI (Smith and Simmons, 2004, 2005) is an approximate value-iteration technique that performs a heuristic search through the belief space for beliefs at which to update the bounds, similar to work by Satia and Lave (1973).

12.4 Decision Making Without a-Priori Models

When no models of the environment are available to the agent *a priori*, the model-based methods presented in the previous section cannot be directly applied. Even relatively simple techniques such as Q_{MDP} (Section 12.3.1) require knowledge of the complete POMDP model: the solution to the underlying MDP is computed using the transition and reward model, while the belief update (12.2) additionally requires the observation model.

In general, there exist two ways of tackling such a decision-making problem, known as direct and indirect reinforcement learning methods. Direct methods apply true model-free techniques, which do not try to reconstruct the unknown POMDP models, but for instance map observation histories directly to actions. On the other extreme, one can attempt to reconstruct the POMDP model by interacting with it, which then in principle can be solved using techniques presented in Section 12.3.

This indirect approach has long been out of favor for POMDPs, as (i) reconstructing (an approximation of) the POMDP models is very hard, and (ii) even with a recovered POMDP, model-based methods would take prohibitively long to compute a good policy. However, advances in model-based methods such as the point-based family of algorithms (Section 12.3.4) have made these types of approaches more attractive.

12.4.1 Memoryless Techniques

First, we consider methods for learning memoryless policies, that is, policies that map each observation that an agent receives directly to an action, without consulting any internal state. Memoryless policies can either be deterministic mappings, $\pi : \Omega \rightarrow A$, or probabilistic mappings, $\pi : \Omega \rightarrow \Delta(A)$. As illustrated by the example in Section 12.2.3, probabilistic policies allow for higher payoffs, at the cost of an increased search space that no longer can be enumerated (Singh et al, 1994). In fact, the problem of finding an optimal deterministic memoryless policy has been shown to be NP-hard (Littman, 1994), while the complexity of determining the optimal probabilistic memoryless policy is still an open problem.

Loch and Singh (1998) have demonstrated empirically that using eligibility traces, in their case in SARSA(λ), can improve the ability of memoryless methods to handle partial observability. SARSA(λ) was shown to learn the optimal deterministic memoryless policy in several domains (for which it was possible to enumerate all such policies, of which there are $|A|^{\Omega}$). Bagnell et al (2004) also consider the memoryless deterministic case, but using non-stationary policies instead of stationary ones. They show that successful non-stationary policies can be found in certain maze domains for which no good stationary policies exist. Regarding learning stochastic memoryless policies, an algorithm has been proposed by Jaakkola et al (1995), and tested empirically by Williams and Singh (1999), showing that it can successfully learn stochastic memoryless policies. An interesting twist is provided by Hierarchical Q-Learning (Wiering and Schmidhuber, 1997), which aims to learn a subgoal sequence in a POMDP, where each subgoal can be successfully achieved using a memoryless policy.

12.4.2 Learning Internal Memory

Given the limitations of memoryless policies in systems without a Markovian state signal such as POMDPs, a natural evolution in research has been to incorporate some form of memory, so-called internal state, in each agent. Storing the complete history of the process, i.e., the vector of actions taken by the agent and observations received, is not a practical option for several reasons. First of all, as in the model-free case the agent is not able to compute a belief state, this representation grows without bounds. A second reason is that such a representation does not allow

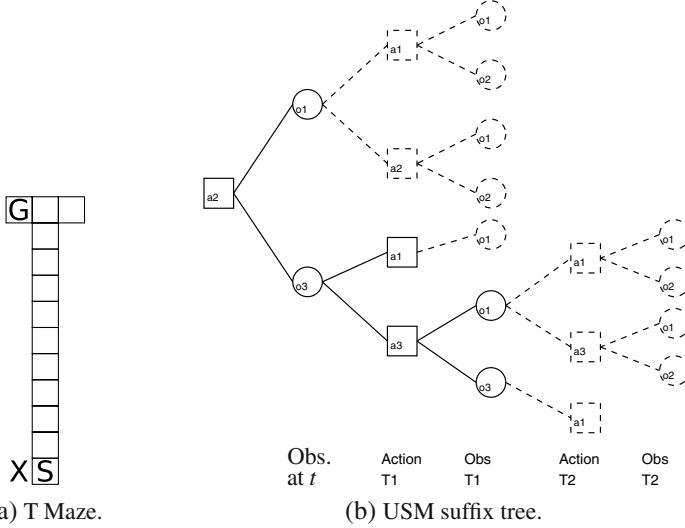


Fig. 12.7 (a) Long-term dependency T maze (Bakker, 2002). (b) Example of a suffix tree used by the USM algorithm (McCallum, 1995), where fringe nodes are indicated by dashed lines.

for easy generalization, e.g., it is not clear how experience obtained after history $\langle a^1, o^1, a^1, o^1 \rangle$ can be used to update the value for history $\langle a^2, o^1, a^1, o^1 \rangle$. To counter these problems, researchers have proposed many different internal-state representations, of which we give a brief overview.

First of all, the memoryless methods presented before can be seen as maintaining a history window of only a single observation. Instead, these algorithms can also be applied with a history window containing the last k observations (Littman, 1994; Loch and Singh, 1998), where k is typically an a-priori defined parameter. In some domains such a relatively cheap increase of the policy space (by means of a low k) can buy a significant improvement in learning time and task performance. Finite history windows have also been used as a representation for neural networks (Lin and Mitchell, 1992).

Finite history windows cannot however capture arbitrary long-term dependencies, such as for instance present in the T Maze in Figure 12.7a, an example provided by Bakker (2002). In this problem the agent starts at S, and needs to navigate to G. However, the location of G is unknown initially, and might be on the left or on the right at the end of the corridor. However, in the start state the agent can observe a road sign X, which depends on the particular goal location. The length of the corridor can be varied (in Figure 12.7a it is 10), meaning that the agent needs to learn to remember the road sign many time steps. Obviously, such a dependency cannot be represented well by finite history windows.

Alleviating the problem of fixed history windows, McCallum (1993, 1995, 1996) proposed several algorithms for variable history windows, among other contributions. These techniques allow for the history window to have a different depth in different parts of the state space. For instance, Utile Suffix Memory (USM) learns a short-term memory representation by growing a suffix tree (McCallum, 1995), an example of which is shown in Figure 12.7b. USM groups together RL experiences based on how much history it considers significant for each instance. In this sense, in different parts of the state space different history lengths can be maintained, in contrast to the finite history window approaches. A suffix tree representation is depicted by solid lines in Figure 12.7b, where the leaves cluster instances that have a matching history up to the corresponding depth. The dashed nodes are the so-called fringe nodes: additional branches in the tree that the algorithm can consider to add to the tree. When a statistical test indicates that instances in a branch of fringe nodes come from different distributions of the expected future discounted reward, the tree is grown to include this fringe branch. Put otherwise, if adding the branch will help predicting the future rewards, it is worthwhile to extend the memory in the corresponding part of the state space. More recent work building on these ideas focuses on better learning behavior in the presence of noisy observations (Shani and Brafman, 2005; Wierstra and Wiering, 2004). Along these lines, recurrent neural networks, for instance based on the Long Short-Term Memory architecture, have also been successfully used as internal state representation (Hochreiter and Schmidhuber, 1997; Bakker, 2002).

Other representations have been proposed as well. Meuleau et al (1999b) extend the VAPS algorithm (Baird and Moore, 1999) to learn policies represented as Finite State Automata (FSA). The FSA represent finite policy graphs, in which nodes are labelled with actions, and the arcs with observations. As in VAPS, stochastic gradient ascent is used to converge to a locally optimal controller. The problem of finding the optimal policy graph of a given size has also been studied (Meuleau et al, 1999a). However, note that the optimal POMDP policy can require an infinite policy graph to be properly represented.

Finally, predictive state representations (PSRs) have been proposed as an alternative to POMDPs for modeling stochastic and partially observable environments (Littman et al, 2002; Singh et al, 2004), see Chapter 13. A PSR dispenses with the hidden POMDP states, and only considers sequences of action and observations which are observed quantities. In a PSR, the state of the system is expressed in possible future event sequences, or “core tests”, of alternating actions and observations. The state of a PSR is defined as a vector of probabilities that each core test can actually be realized, given the current history. The advantages of PSRs are most apparent in model-free learning settings, as the model only considers observable events instead of hidden states.

12.5 Recent Trends

To conclude, we discuss some types of approaches that have been gaining popularity recently.

Most of the model-based methods discussed in this chapter are offline techniques that determine *a priori* what action to take in each situation the agent might encounter. Online approaches, on the other hand, only compute what action to take at the current moment (Ross et al, 2008b). Focusing exclusively on the current decision can provide significant computational savings in certain domains, as the agent does not have to plan for areas of the state space which it never encounters. However, the need to choose actions every time step implies severe constraints on the online search time. Offline point-based methods can be used to compute a rough value function, serving as the online search heuristic. In a similar manner, Monte Carlo approaches are also appealing for large POMDPs, as they only require a generative model (black box simulator) to be available and they have the potential to mitigate the curse of dimensionality (Thrun, 2000; Kearns et al, 2000; Silver and Veness, 2010).

As discussed in detail in Chapter 11, Bayesian RL techniques are promising for POMDPs, as they provide an integrated way of exploring and exploiting models. Put otherwise, they do not require interleaving the model-learning phases (e.g., using Baum-Welch (Koenig and Simmons, 1996) or other methods (Shani et al, 2005)) with model-exploitation phases, which could be a naive approach to apply model-based methods to unknown POMDPs. Poupart and Vlassis (2008) extended the BEETLE algorithm (Poupart et al, 2006), a Bayesian RL method for MDPs, to partially observable settings. As other Bayesian RL methods, the models are represented by Dirichlet distributions, and learning involves updating the Dirichlet hyperparameters. The work is more general than the earlier work by Jaulmes et al (2005), which required the existence of an oracle that the agent could query to reveal the true state. Ross et al (2008a) proposed the Bayes-Adaptive POMDP model, an alternative model for Bayesian reinforcement learning which extends Bayes-Adaptive MDPs (Duff, 2002). All these methods assume that the size of the state, observation and action spaces are known.

Policy gradient methods search in a space of parameterized policies, optimizing the policy by performing gradient ascent in the parameter space (Peters and Bagnell, 2010). As these methods do not require to estimate a belief state (Aberdeen and Baxter, 2002), they have been readily applied in POMDPs, with impressive results (Peters and Schaal, 2008).

Finally, a recent trend has been to cast the model-based RL problem as one of probabilistic inference, for instance using Expectation Maximization for computing optimal policies in MDPs. Vlassis and Toussaint (2009) showed how such methods can also be extended to the model-free POMDP case. In general, inference methods can provide fresh insights in well-known RL algorithms.

Acknowledgements. This work was funded by Fundação para a Ciência e a Tecnologia (ISR/IST pluriannual funding) through the PIDDAC Program funds and was supported by project PTDC/EEA-ACR/73266/2006.

References

- Aberdeen, D., Baxter, J.: Scaling internal-state policy-gradient methods for POMDPs. In: International Conference on Machine Learning (2002)
- Åström, K.J.: Optimal control of Markov processes with incomplete state information. Journal of Mathematical Analysis and Applications 10(1), 174–205 (1965)
- Bagnell, J.A., Kakade, S., Ng, A.Y., Schneider, J.: Policy search by dynamic programming. In: Advances in Neural Information Processing Systems, vol. 16. MIT Press (2004)
- Baird, L., Moore, A.: Gradient descent for general reinforcement learning. In: Advances in Neural Information Processing Systems, vol. 11. MIT Press (1999)
- Bakker, B.: Reinforcement learning with long short-term memory. In: Advances in Neural Information Processing Systems, vol. 14. MIT Press (2002)
- Baxter, J., Bartlett, P.L.: Infinite-horizon policy-gradient estimation. Journal of Artificial Intelligence Research 15, 319–350 (2001)
- Baxter, J., Bartlett, P.L., Weaver, L.: Experiments with infinite-horizon, policy-gradient estimation. Journal of Artificial Intelligence Research 15, 351–381 (2001)
- Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The complexity of decentralized control of Markov decision processes. Mathematics of Operations Research 27(4), 819–840 (2002)
- Bonet, B.: An epsilon-optimal grid-based algorithm for partially observable Markov decision processes. In: International Conference on Machine Learning (2002)
- Boutilier, C., Poole, D.: Computing optimal policies for partially observable decision processes using compact representations. In: Proc. of the National Conference on Artificial Intelligence (1996)
- Brafman, R.I.: A heuristic variable grid solution method for POMDPs. In: Proc. of the National Conference on Artificial Intelligence (1997)
- Braziuas, D., Boutilier, C.: Stochastic local search for POMDP controllers. In: Proc. of the National Conference on Artificial Intelligence (2004)
- Brunskill, E., Kaelbling, L., Lozano-Perez, T., Roy, N.: Continuous-state POMDPs with hybrid dynamics. In: Proc. of the Int. Symposium on Artificial Intelligence and Mathematics (2008)
- Cassandra, A.R.: Exact and approximate algorithms for partially observable Markov decision processes. PhD thesis, Brown University (1998)
- Cassandra, A.R., Kaelbling, L.P., Littman, M.L.: Acting optimally in partially observable stochastic domains. In: Proc. of the National Conference on Artificial Intelligence (1994)
- Cassandra, A.R., Kaelbling, L.P., Kurien, J.A.: Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In: Proc. of International Conference on Intelligent Robots and Systems (1996)
- Cassandra, A.R., Littman, M.L., Zhang, N.L.: Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In: Proc. of Uncertainty in Artificial Intelligence (1997)
- Cheng, H.T.: Algorithms for partially observable Markov decision processes. PhD thesis, University of British Columbia (1988)

- Doshi, F., Roy, N.: The permutable POMDP: fast solutions to POMDPs for preference elicitation. In: Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems (2008)
- Drake, A.W.: Observation of a Markov process through a noisy channel. Sc.D. thesis, Massachusetts Institute of Technology (1962)
- Duff, M.: Optimal learning: Computational procedures for Bayes-adaptive Markov decision processes. PhD thesis, University of Massachusetts, Amherst (2002)
- Dynkin, E.B.: Controlled random sequences. *Theory of Probability and its Applications* 10(1), 1–14 (1965)
- Ellis, J.H., Jiang, M., Corotis, R.: Inspection, maintenance, and repair with partial observability. *Journal of Infrastructure Systems* 1(2), 92–99 (1995)
- Feng, Z., Zilberstein, S.: Region-based incremental pruning for POMDPs. In: Proc. of Uncertainty in Artificial Intelligence (2004)
- Foka, A., Trahanias, P.: Real-time hierarchical POMDPs for autonomous robot navigation. *Robotics and Autonomous Systems* 55(7), 561–571 (2007)
- Fox, D., Burgard, W., Thrun, S.: Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research* 11, 391–427 (1999)
- Haight, R.G., Polasky, S.: Optimal control of an invasive species with imperfect information about the level of infestation. *Resource and Energy Economics* (2010) (in Press, Corrected Proof)
- Hansen, E.A.: Finite-memory control of partially observable systems. PhD thesis, University of Massachusetts, Amherst (1998a)
- Hansen, E.A.: Solving POMDPs by searching in policy space. In: Proc. of Uncertainty in Artificial Intelligence (1998b)
- Hansen, E.A., Feng, Z.: Dynamic programming for POMDPs using a factored state representation. In: Int. Conf. on Artificial Intelligence Planning and Scheduling (2000)
- Hauskrecht, M.: Value function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research* 13, 33–95 (2000)
- Hauskrecht, M., Fraser, H.: Planning treatment of ischemic heart disease with partially observable Markov decision processes. *Artificial Intelligence in Medicine* 18, 221–244 (2000)
- Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Computation* 9(8), 1735–1780 (1997)
- Hoey, J., Little, J.J.: Value-directed human behavior analysis from video using partially observable Markov decision processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29(7), 1–15 (2007)
- Hoey, J., Poupart, P.: Solving POMDPs with continuous or large discrete observation spaces. In: Proc. Int. Joint Conf. on Artificial Intelligence (2005)
- Hsiao, K., Kaelbling, L., Lozano-Perez, T.: Grasping pomdps. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, pp. 4685–4692 (2007)
- Jaakkola, T., Singh, S.P., Jordan, M.I.: Reinforcement learning algorithm for partially observable Markov decision problems. In: Advances in Neural Information Processing Systems, vol. 7 (1995)
- Jaulmes, R., Pineau, J., Precup, D.: Active Learning in Partially Observable Markov Decision Processes. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 601–608. Springer, Heidelberg (2005)
- Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101, 99–134 (1998)

- Kearns, M., Mansour, Y., Ng, A.Y.: Approximate planning in large POMDPs via reusable trajectories. In: Advances in Neural Information Processing Systems, vol. 12. MIT Press (2000)
- Koenig, S., Simmons, R.: Unsupervised learning of probabilistic models for robot navigation. In: Proc. of the IEEE Int. Conf. on Robotics and Automation (1996)
- Kurniawati, H., Hsu, D., Lee, W.: SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Robotics: Science and Systems (2008)
- Lin, L., Mitchell, T.: Memory approaches to reinforcement learning in non-Markovian domains. Tech. rep., Carnegie Mellon University, Pittsburgh, PA, USA (1992)
- Lin, Z.Z., Bean, J.C., White, C.C.: A hybrid genetic/optimization algorithm for finite horizon, partially observed Markov decision processes. INFORMS Journal on Computing 16(1), 27–38 (2004)
- Littman, M.L.: Memoryless policies: theoretical limitations and practical results. In: Proc. of the 3rd Int. Conf. on Simulation of Adaptive Behavior: from Animals to Animats 3, pp. 238–245. MIT Press, Cambridge (1994)
- Littman, M.L.: Algorithms for sequential decision making. PhD thesis, Brown University (1996)
- Littman, M.L., Cassandra, A.R., Kaelbling, L.P.: Learning policies for partially observable environments: Scaling up. In: International Conference on Machine Learning (1995)
- Littman, M.L., Sutton, R.S., Singh, S.: Predictive representations of state. In: Advances in Neural Information Processing Systems, vol. 14. MIT Press (2002)
- Loch, J., Singh, S.: Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In: International Conference on Machine Learning (1998)
- Lovejoy, W.S.: Computationally feasible bounds for partially observed Markov decision processes. Operations Research 39(1), 162–175 (1991)
- Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and related stochastic optimization problems. Artificial Intelligence 147(1-2), 5–34 (2003)
- McCallum, R.A.: Overcoming incomplete perception with utile distinction memory. In: International Conference on Machine Learning (1993)
- McCallum, R.A.: Instance-based utile distinctions for reinforcement learning with hidden state. In: International Conference on Machine Learning (1995)
- McCallum, R.A.: Reinforcement learning with selective perception and hidden state. PhD thesis, University of Rochester (1996)
- Meuleau, N., Kim, K.E., Kaelbling, L.P., Cassandra, A.R.: Solving POMDPs by searching the space of finite policies. In: Proc. of Uncertainty in Artificial Intelligence (1999a)
- Meuleau, N., Peshkin, L., Kim, K.E., Kaelbling, L.P.: Learning finite-state controllers for partially observable environments. In: Proc. of Uncertainty in Artificial Intelligence (1999b)
- Monahan, G.E.: A survey of partially observable Markov decision processes: theory, models and algorithms. Management Science 28(1) (1982)
- Ng, A.Y., Jordan, M.: PEGASUS: A policy search method for large MDPs and POMDPs. In: Proc. of Uncertainty in Artificial Intelligence (2000)
- Oliehoek, F.A., Spaan, M.T.J., Vlassis, N.: Optimal and approximate Q-value functions for decentralized POMDPs. Journal of Artificial Intelligence Research 32, 289–353 (2008)
- Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of Markov decision processes. Mathematics of Operations Research 12(3), 441–450 (1987)
- Parr, R., Russell, S.: Approximating optimal policies for partially observable stochastic domains. In: Proc. Int. Joint Conf. on Artificial Intelligence (1995)

- Peters, J., Bagnell, J.A.D.: Policy gradient methods. In: Springer Encyclopedia of Machine Learning. Springer, Heidelberg (2010)
- Peters, J., Schaal, S.: Natural actor-critic. Neurocomputing 71, 1180–1190 (2008)
- Pineau, J., Thrun, S.: An integrated approach to hierarchy and abstraction for POMDPs. Tech. Rep. CMU-RI-TR-02-21, Robotics Institute, Carnegie Mellon University (2002)
- Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for POMDPs. In: Proc. Int. Joint Conf. on Artificial Intelligence (2003)
- Platzman, L.K.: A feasible computational approach to infinite-horizon partially-observed Markov decision problems. Tech. Rep. J-81-2, School of Industrial and Systems Engineering, Georgia Institute of Technology, reprinted in working notes AAAI, Fall Symposium on Planning with POMDPs (1981)
- Poon, K.M.: A fast heuristic algorithm for decision-theoretic planning. Master's thesis, The Hong-Kong University of Science and Technology (2001)
- Porta, J.M., Spaan, M.T.J., Vlassis, N.: Robot planning in partially observable continuous domains. In: Robotics: Science and Systems (2005)
- Porta, J.M., Vlassis, N., Spaan, M.T.J., Poupart, P.: Point-based value iteration for continuous POMDPs. Journal of Machine Learning Research 7, 2329–2367 (2006)
- Poupart, P.: Exploiting structure to efficiently solve large scale partially observable Markov decision processes. PhD thesis, University of Toronto (2005)
- Poupart, P., Boutilier, C.: Bounded finite state controllers. In: Advances in Neural Information Processing Systems, vol. 16. MIT Press (2004)
- Poupart, P., Vlassis, N.: Model-based Bayesian reinforcement learning in partially observable domains. In: International Symposium on Artificial Intelligence and Mathematics, ISAIM (2008)
- Poupart, P., Vlassis, N., Hoey, J., Regan, K.: An analytic solution to discrete Bayesian reinforcement learning. In: International Conference on Machine Learning (2006)
- Ross, S., Chaib-draa, B., Pineau, J.: Bayes-adaptive POMDPs. In: Advances in Neural Information Processing Systems, vol. 20, pp. 1225–1232. MIT Press (2008a)
- Ross, S., Pineau, J., Paquet, S., Chaib-draa, B.: Online planning algorithms for POMDPs. Journal of Artificial Intelligence Research 32, 664–704 (2008b)
- Roy, N., Gordon, G.: Exponential family PCA for belief compression in POMDPs. In: Advances in Neural Information Processing Systems, vol. 15. MIT Press (2003)
- Roy, N., Thrun, S.: Coastal navigation with mobile robots. In: Advances in Neural Information Processing Systems, vol. 12. MIT Press (2000)
- Roy, N., Gordon, G., Thrun, S.: Finding approximate POMDP solutions through belief compression. Journal of Artificial Intelligence Research 23, 1–40 (2005)
- Sanner, S., Kersting, K.: Symbolic dynamic programming for first-order POMDPs. In: Proc. of the National Conference on Artificial Intelligence (2010)
- Satia, J.K., Lave, R.E.: Markovian decision processes with probabilistic observation of states. Management Science 20(1), 1–13 (1973)
- Seuken, S., Zilberstein, S.: Formal models and algorithms for decentralized decision making under uncertainty. Autonomous Agents and Multi-Agent Systems (2008)
- Shani, G., Brafman, R.I.: Resolving perceptual aliasing in the presence of noisy sensors. In: Saul, L.K., Weiss, Y., Bottou, L. (eds.) Advances in Neural Information Processing Systems, vol. 17, pp. 1249–1256. MIT Press, Cambridge (2005)
- Shani, G., Brafman, R.I., Shimony, S.E.: Model-Based Online Learning of POMDPs. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 353–364. Springer, Heidelberg (2005)

- Shani, G., Brafman, R.I., Shimony, S.E.: Forward search value iteration for POMDPs. In: Proc. Int. Joint Conf. on Artificial Intelligence (2007)
- Shani, G., Poupart, P., Brafman, R.I., Shimony, S.E.: Efficient ADD operations for point-based algorithms. In: Int. Conf. on Automated Planning and Scheduling (2008)
- Silver, D., Veness, J.: Monte-carlo planning in large POMDPs. In: Lafferty, J., Williams, C.K.I., Shawe-Taylor, J., Zemel, R., Culotta, A. (eds.) Advances in Neural Information Processing Systems, vol. 23, pp. 2164–2172 (2010)
- Simmons, R., Koenig, S.: Probabilistic robot navigation in partially observable environments. In: Proc. Int. Joint Conf. on Artificial Intelligence (1995)
- Singh, S., Jaakkola, T., Jordan, M.: Learning without state-estimation in partially observable Markovian decision processes. In: International Conference on Machine Learning (1994)
- Singh, S., James, M.R., Rudary, M.R.: Predictive state representations: A new theory for modeling dynamical systems. In: Proc. of Uncertainty in Artificial Intelligence (2004)
- Smallwood, R.D., Sondik, E.J.: The optimal control of partially observable Markov decision processes over a finite horizon. Operations Research 21, 1071–1088 (1973)
- Smith, T., Simmons, R.: Heuristic search value iteration for POMDPs. In: Proc. of Uncertainty in Artificial Intelligence (2004)
- Smith, T., Simmons, R.: Point-based POMDP algorithms: Improved analysis and implementation. In: Proc. of Uncertainty in Artificial Intelligence (2005)
- Sondik, E.J.: The optimal control of partially observable Markov processes. PhD thesis, Stanford University (1971)
- Spaan, M.T.J., Vlassis, N.: A point-based POMDP algorithm for robot planning. In: Proc. of the IEEE Int. Conf. on Robotics and Automation (2004)
- Spaan, M.T.J., Vlassis, N.: Perseus: Randomized point-based value iteration for POMDPs. Journal of Artificial Intelligence Research 24, 195–220 (2005a)
- Spaan, M.T.J., Vlassis, N.: Planning with continuous actions in partially observable environments. In: Proc. of the IEEE Int. Conf. on Robotics and Automation (2005b)
- Spaan, M.T.J., Veiga, T.S., Lima, P.U.: Active cooperative perception in network robot systems using POMDPs. In: Proc. of International Conference on Intelligent Robots and Systems (2010)
- Sridharan, M., Wyatt, J., Dearden, R.: Planning to see: A hierarchical approach to planning visual actions on a robot using POMDPs. Artificial Intelligence 174, 704–725 (2010)
- Stankiewicz, B., Cassandra, A., McCabe, M., Weathers, W.: Development and evaluation of a Bayesian low-vision navigation aid. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 37(6), 970–983 (2007)
- Stratonovich, R.L.: Conditional Markov processes. Theory of Probability and Its Applications 5(2), 156–178 (1960)
- Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
- Theocharous, G., Mahadevan, S.: Approximate planning with hierarchical partially observable Markov decision processes for robot navigation. In: Proc. of the IEEE Int. Conf. on Robotics and Automation (2002)
- Thrun, S.: Monte Carlo POMDPs. In: Advances in Neural Information Processing Systems, vol. 12. MIT Press (2000)
- Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. MIT Press (2005)
- Varakantham, P., Maheswaran, R., Tambe, M.: Exploiting belief bounds: Practical POMDPs for personal assistant agents. In: Proc. of Int. Conference on Autonomous Agents and Multi Agent Systems (2005)
- Vlassis, N., Toussaint, M.: Model-free reinforcement learning as mixture learning. In: International Conference on Machine Learning, pp. 1081–1088. ACM (2009)

- Wang, C., Kharden, R.: Relational partially observable MDPs. In: Proc. of the National Conference on Artificial Intelligence (2010)
- White, C.C.: Partially observed Markov decision processes: a survey. *Annals of Operations Research* 32 (1991)
- Wiering, M., Schmidhuber, J.: HQ-learning. *Adaptive Behavior* 6(2), 219–246 (1997)
- Wierstra, D., Wiering, M.: Utile distinction hidden Markov models. In: International Conference on Machine Learning (2004)
- Williams, J.D., Young, S.: Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language* 21(2), 393–422 (2007)
- Williams, J.K., Singh, S.: Experimental results on learning stochastic memoryless policies for partially observable Markov decision processes. In: Advances in Neural Information Processing Systems, vol. 11 (1999)
- Zhang, N.L., Liu, W.: Planning in stochastic domains: problem characteristics and approximations. Tech. Rep. HKUST-CS96-31, Department of Computer Science, The Hong Kong University of Science and Technology (1996)
- Zhou, R., Hansen, E.A.: An improved grid-based approximation algorithm for POMDPs. In: Proc. Int. Joint Conf. on Artificial Intelligence (2001)

Chapter 13

Predictively Defined Representations of State

David Wingate

Abstract. The concept of *state* is central to dynamical systems. In any timeseries problem—such as filtering, planning or forecasting—models and algorithms summarize important information from the past into some sort of state variable. In this chapter, we start with a broad examination of the concept of state, with emphasis on the fact that there are many possible representations of state for a given dynamical system, each with different theoretical and computational properties. We then focus on *models with predictively defined representations of state* that represent state as a set of statistics about the short-term future, as opposed to the classic approach of treating state as a latent, unobservable quantity. In other words, the past is summarized into predictions about the actions and observations in the short-term future, which can be used to make further predictions about the infinite future. While this representational idea applies to any dynamical system problem, it is particularly useful in a model-based RL context, when an agent must learn a representation of state and a model of system dynamics online: because the representation (and hence all of the model’s parameters) are defined using only statistics of observable quantities, their learning algorithms are often straightforward and have attractive theoretical properties. Here, we survey the basic concepts of predictively defined representations of state, important auxiliary constructs (such as the systems dynamics matrix), and theoretical results on their representational power and learnability.

Parts of this chapter were originally published as *Exponential Family Predictive Representations of State* by David Wingate (2008).

David Wingate
Massachusetts Institute of Technology, Cambridge, MA, 02139
e-mail: wingated@mit.edu

13.1 Introduction

This chapter considers the class of dynamical systems models with *predictively defined representations of state*. To motivate and introduce the idea, we will first re-examine the fundamental notion of state, why it is important to an agent, and why different representations of state might have different theoretical or computational properties—particularly as they apply to learning both a representation and a system dynamics model online.

In the model-based RL setting, for example, an agent must learn a model from data acquired from interactions with an environment. These models are used with planning algorithms to help an agent choose optimal actions by reasoning about the future, so the fundamental capability models must have is to predict the consequences of actions in terms of future observations and rewards. To make the best predictions possible, models must summarize important past experience. Such a summary of the past is known as *state*, which we will define formally momentarily. For now, we informally define state as a summary of an agent’s knowledge about the state of affairs in the environment. In Markov Decision Processes (MDPs), for example, the most recent observation constitutes state, meaning that building a model simply involves learning transitions between states. In partially observable domains, however, the most recent observation does not constitute state, so an agent must learn what is important about the past, how to remember it, and how to use it to predict the future.

This chapter examines in-depth a particular class of state representations known as “predictively defined representations of state”. In a model with a predictively defined representation of state, an agent summarizes the past as a set of predictions about the short-term future that allow the agent to make further predictions about the infinite future. This representation often seems counter-intuitive at first, but the idea is simple: every state is a summary of a past, and every past implies a distribution over the future—so why not represent state directly as sufficient statistics of this distribution over possible futures?

Here, we will examine the theoretical foundations of this type of representation, and will compare their computational and representational characteristics with more traditional notions of state. We will focus on how they can be learned directly from data acquired online, by considering two key parts of model building: learning a representation of state, and learning an algorithm which allows an agent to accurately maintain state in response to new information.

13.1.1 What Is “State”?

This chapter is intimately concerned with the idea of state and how it can be represented. We now define state more precisely, with emphasis on the fact that there are multiple acceptable representations of state. Throughout this chapter, we will always discuss state from the perspective of the agent, as opposed to that of an

omniscient observer—a perspective motivated from our focus on agents who must learn a model.

What exactly is “state”? Informally, state is the current situation that the agent finds itself in: for a robot, state might be the position and angle of all its actuators, as well as its map coordinates, battery status, the goal it is trying to achieve, etc. In a Markov system, all of these variables are given to the agent. However, in a partially observable system the agent may not have immediate access to all of the information that it would like to know about its situation: a robot with a broken sensor may not know the exact position of its arm; a stock trader may not know exactly what the long-term strategies are of all the companies he is investing in; and a baseball batter may not know exactly how fast the baseball is approaching.

In partially observable domains, there is a more formal definition of state: state is a summary of all of the information that an agent could possibly have about its current situation. This information is contained in the history of actions and observations it has experienced, which means that a full history constitutes perfect state and is therefore the standard against which other state representations are measured. Of course, an agent will usually want to compress this history—without losing information!—because otherwise it must store an increasingly large amount of information as it interacts with the environment longer and longer. This motivates our formal definition of state:

State is any finite-dimensional statistic of history which is sufficient to predict the distribution of future rewards and observations.

We will sometimes abbreviate this by simply saying that “state is a sufficient statistic for history.”

Defining state in this way implies an important conditional independence: that the distribution over future observations is conditionally independent of the past, given state:

$$p(\text{future}|\text{state,past}) = p(\text{future}|\text{state}).$$

This means that since state has summarized all of the information in a history which is relevant for predicting the future, we may discard the history itself. As it turns out, this summary of history is also sufficient for the agent to act optimally (Astrom, 1965). Thus there is a close connection between representing state, maintaining state, summarizing the past, predicting the future and acting optimally.

Unfortunately, there are some conflicting uses for the word “state.” In POMDPs, for example, the model posits underlying “states” which are assumed to represent the “true” state of the process, which is unobserved by the agent (when necessary, we will refer to these underlying states as “latent states”). In POMDP terminology, the agent summarizes a history with a “belief state,” which is a distribution over latent states. According to our definition, it is the belief state which is the sufficient statistic for history. However, latent states are not essential for sufficiency; as we shall see, one of the contributions of this chapter is the introduction of numerous concepts of state which make no reference to any sort of latent state.

13.1.2 Which Representation of State?

There are many acceptable summaries of history. To illustrate this, consider the example of a robot localization problem. A small robot wanders through a building, and must track its position, but it is given only a camera sensor. There are many possible representations for the robot’s position. Its pose could be captured in terms of a distribution over x,y coordinates, for example. However, it could also be described in terms of, say, a distribution over polar coordinates. Cartesian and polar coordinates are different representations of state which are equally expressive, but *both* are internal to the agent. From the perspective of the environment, neither is more accurate, or more correct, or more useful—and we have said nothing about how either Cartesian or polar coordinates could be accurately maintained given nothing but camera images. It is easy to see that there are an infinite number of such state representations: any one-to-one transformation of state is still state, and adding redundant information to state is still state.

The realization that there are multiple satisfactory representations of state begs the question: among all possible concepts of state, why should one be preferred over another? There are many criteria that could be used to compare competing representations. For example, a representation might be preferred if:

- It is easier for a human (designing an agent) to understand and use.
- It somehow “matches” another agent’s notion of state.
- It has favorable computational or statistical properties.

Not every statistic of history will be sufficient for predicting the future, which means that some representations may only constitute approximate state. Among approximately sufficient representations, one might be preferred if:

- It is more expressive than another representation.
- It is less expressive than another, but is still sufficient to do what we want to do with it (for example, control the system optimally).

Thus, even among state representations which are equally expressive, there might be reasons to prefer one over another.

Because we are interested in learning agents, we are interested in *learnable* representation of state—those for which effective learning algorithms are available. The idea that one representation of state may be more learnable than another motivates our first distinction between different representations of state: *grounded*¹ representations of state are those in which every component of the state is defined using only statistics about observable quantities (which could be either observables in the future, or the past), and *latent* representations of state refer to everything else. In our robot localization example, both Cartesian and polar coordinates are latent representations of state, because neither is explicitly observed by the robot; only a state representation defined in terms of features of camera images could be defined as grounded.

¹ Some disciplines may have other definitions of the word “grounded” which are specific and technical; we avoid them.

Within the class of grounded representations of state, we will make further distinctions. Some grounded representations may be defined in terms of past observations, as in the case of k -th order Markov models (where the past k observations constitute state), and others could be defined in terms of the current observation.

There is also a third class of grounded representations, which is the subject of this chapter: *predictively defined representations of state*. In a predictively defined representation of state, state is represented as statistics about features of *future* observations. These statistics are flexible: they may be the parameters of a distribution over the short-term future, represent the expectations of random variables in the future, represent the densities of specific futures, represent statements about future strings of observations given possible future actions, etc. It is this class of state representations which we will investigate throughout the chapter, along with algorithms for learning and maintaining that state.

13.1.3 Why Predictions about the Future?

Why investigate models with predictively defined representations of state? We are motivated for three reasons:

- **Learnability.** The central problem this chapter addresses is learning models of dynamical systems from data. The fact that all of the parameters of predictively defined models have direct, statistical relationships with observable quantities suggests that predictively defined models may be more learnable than classical counterparts. As an example of this, the parameter estimation algorithm of Rudary et al (2005) for the Predictive Linear Gaussian (PLG) model (the predictively defined version of the Kalman filter, discussed later) was shown to be statistically consistent, which is a strong learning guarantee.
- **Representational Ability.** To date, most models with predictively defined state have been shown to be at least as expressive as their classical counterpart. For example, PLGs are as expressive as Kalman filters (Rudary et al, 2005), and linear PSRs (discussed in Section 13.2) are strictly more expressive than POMDPs (James, 2005). That is, there are domains which can be modeled by a finite PSR which cannot be modeled by any finite POMDP, but every finite POMDP can be modeled by a finite PSR. This representational ability is achieved without sacrificing compactness: linear PSRs are never larger than their equivalent POMDPs, PLGs are never larger than their equivalent Kalman filters, and there are examples of PSRs which have an exponentially smaller number of parameters than their equivalent POMDP (Littman et al, 2002). For a given data set, a model with fewer parameters is likely to have greater statistical efficiency, which is useful for learnability.
- **Generalization.** As a knowledge representation, predictions about the future may have attractive properties from the perspective of function approximation

and generalization. For example, Rafols et al (2005) have provided some preliminary evidence that predictive representations provide a better basis for generalization than latent ones. To see the intuition for this, consider the problem of assigning values to states for a domain in which an agent must navigate a maze. Using a predictively defined representation, two states are “near” each other when their distributions over the future are similar; if that is true, it is likely that they should be assigned similar values. But if the agent uses, say, Cartesian coordinates as a state representation, two states which are nearby in Euclidean space may not necessarily have similar values. The classic example is two states on either side of a wall: although the two states appear to be close, an agent may have to travel long distances through the maze to reach one from the other, and they should be assigned different values, which may be difficult to do with a smooth function approximator. Littman et al (2002) have also suggested that in compositional domains, predictions could also be useful in learning to make other predictions, stating that in many cases “the solutions to earlier [prediction] problems have been shown to provide features that generalize particularly well to subsequent [prediction] problems.” This was also partly demonstrated in Izadi and Precup (2005), who showed that PSRs have a natural ability to compress symmetries in state spaces.

The rest of this chapter surveys important concepts and results for models with predictively defined representations of state. Section 13.2 introduces PSRs, Section 13.3 discusses how to learn a PSR model from data, and Section 13.4 discusses planning in PSRs. Section 13.5 tours extensions of PSRs, and Section 13.6 surveys other models with predictively defined state. Finally, Section 13.7 presents some concluding thoughts.

13.2 PSRs

POMDPs have latent states at their heart—a POMDP begins by describing a latent state space, transitions between those states, and the observations that they generate; it defines its sufficient statistic for history as a distribution over these latent states, etc. This model is convenient in several situations, such as when a human designer knows that there really *are* latent states in the system and knows something about their relationships. However, numerous authors have pointed out that while a POMDP is easy to write down, it is notoriously hard to *learn* from data (Nikovski, 2002; Shatkay and Kaelbling, 2002), which is our central concern.

In this section, we turn to alternative model of controlled dynamical systems with discrete observations, called a “Predictive State Representation” (or PSR)². The PSR was introduced by Littman et al (2002), and is one of the first models with a predictively defined representation of state. A PSR is a model capable of capturing dynamical systems that have discrete actions and discrete observations,

² Unfortunately, this name would be more appropriate as a name for an entire class of models.

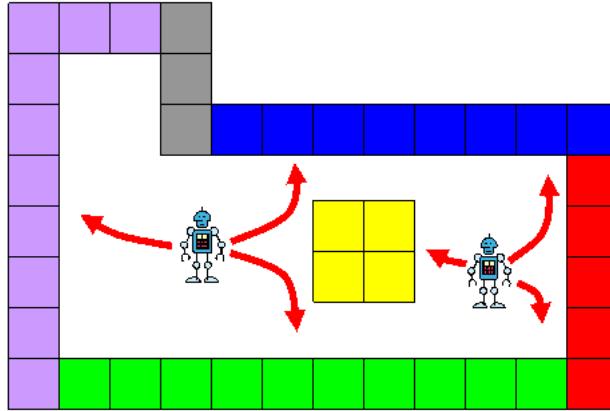


Fig. 13.1 An example of state as predictions about the future. A latent state representation might be defined as x,y coordinates, but a predictively defined state is defined in terms of future possibilities.

like a POMDP. The important contrast is that while POMDPs are built around latent states, PSRs never make any reference to a latent state; instead, a PSR represents state as a set of statistics about the future. This will have positive consequences for learning, as we will see, but importantly, we will lose nothing in terms of modeling capacity: we will see that PSRs can model any system that a finite-state POMDP can, and that many POMDP planning algorithms are directly applicable to PSRs.

We will now introduce the terminology needed to explain PSRs.

13.2.1 Histories and Tests

A *history* is defined as a sequence of actions and observations $a_1 o_1 a_2 o_2 \cdots a_m o_m$ that occurred in the past. A *test* is defined as a possible sequence of future actions and observations $a^1 o^1 a^2 o^2 \cdots a^n o^n$ in the future. An agent is not obligated to take the actions defined in a test—it merely represents one possible future that the agent may wish to reason about.

Figure 13.1 illustrates the idea of tests. The figure shows two robots in a brightly colored maze. In this domain, the actions include “move-left,” “move-right,” etc., and the observations include “bump,” “see pink wall,” “see blue wall,” etc. For the robot on the left, there is a certain action-conditional distribution over the future: if it moves left, it will bump into a pink wall; if it goes to the right then up, it will bump into a blue wall, and if it goes to the right and then down, it will bump into a green wall. The robot on the right is in a different position in the maze, and therefore there is a different distribution over future possibilities: for him, going left results in bumping into a yellow wall, instead of a blue wall. In this simple example, we

can imagine that a sufficiently detailed set of sufficiently long predictions could disambiguate any two positions in the maze—while only referring to observable quantities. This is precisely the intuition behind the state representation in PSRs.

13.2.2 Prediction of a Test

Tests form a central part of the state representation used by PSRs. They are also central to the mathematical objects that PSRs rely on for theoretical results, as well as most learning algorithms for PSRs. This is because from any given history, there is some distribution over possible future sequences of actions and observations that can be captured through the use of tests: each possible future corresponds to a different test, and there is some (possibly zero) probability that each test will occur from every history.

We now define the *prediction* for a test $t = a^1 o^1 a^2 o^2 \cdots a^n o^n$. We assume that the agent is starting in history $h = a_1 o_1 a_2 o_2 \cdots a_m o_m$, meaning that it has taken the actions specified in h and seen the observations specified in h . The prediction of test t is defined to be the probability that the next n observations are exactly those in t , given that the next n actions taken are exactly those in t :

$$p(t|h) = \Pr(o_{m+1} = o^1, o_{m+2} = o^2, \dots, o_{m+n} = o^n | h, a_{m+1} = a^1, \dots, a_{m+n} = a^n).$$

The actions in the test are executed in an open-loop way, without depending on any future information, so $\Pr(a_n | a_1, o_1, \dots, a_{n-1}, o_{n-1}) = \Pr(a_n | a_1, \dots, a_n)$. See (Bowling et al, 2006) for an important discussion of why this is important, particularly when empirically estimating test probabilities.

For ease of notation, we use the following shorthand: for a set of tests $T = \{t_1, t_2, \dots, t_n\}$, the quantity $p(T|h) = [p(t_1|h), p(t_2|h), \dots, p(t_n|h)]^\top$ is a column vector of predictions of the tests in the set T .

13.2.3 The System Dynamics Vector

The *systems dynamics vector* (Singh et al, 2004) is a conceptual construct introduced to define PSRs. This vector describes the evolution of a dynamical system over time: every possible test t has an entry in this vector, representing $p(t|\emptyset)$, or the prediction of t from the null history. These tests are conventionally arranged in length-lexicographic order, from shortest to longest. In general, this is an infinitely long vector, but will still be useful from a theoretical perspective. Here, we will use the notation $a_i^m o_i^n$ to denote the m -th action and the n -th observation at time i :

$$V = [p(a_1^1 o_1^1 | \emptyset), p(a_1^1 o_1^2 | \emptyset), \dots, p(a_1^m o_1^n | \emptyset), p(a_1^1 o_1^1 a_2^1 o_2^1 | \emptyset), \dots]$$

Longer futures →

\mathcal{D}	$a_1^1 o_1^1$	$a_1^1 o_1^2$...	$a_1^m o_1^n$	$a_1^1 o_1^1 a_2^1 o_2^1$...
\emptyset						
$a_1^1 o_1^1$						
\vdots						
$a_1^m o_1^n$				$p(t_i h_j)$		
$a_1^1 o_1^1 a_2^1 o_2^1$						
\vdots						..

↓ Longer histories

Prediction of a test: $p(a_1^m o_1^n | a_1^m o_1^n)$

Fig. 13.2 The system dynamics matrix

The system dynamics vector is representation-independent: *every* dynamical system with discrete observations—including controlled, partially observable, nonstationary, ergodic or any other type—can be completely characterized by a system dynamics vector that makes no reference to latent states of any sort.

13.2.4 The System Dynamics Matrix

The system dynamics matrix D (shown in Figure 13.2) is an important theoretical construct for analyzing properties of histories and tests, and serves as a central object in many PSR learning algorithms.

The system dynamics matrix is obtained by conditioning the system dynamics vector V on all possible histories. In this matrix, the first row is the system dynamics vector, and corresponds to the predictions of every test from the null history. Every possible history has a row in the matrix, and the entries in that row are obtained by conditioning the system dynamics vector on that particular history. An entry i,j in the matrix is the prediction of test j from history i :

$$D_{ij} = p(t_j|h_i) = \frac{p(h_i t_j | \emptyset)}{p(h_i | \emptyset)}.$$

Note that the system dynamics vector has all of the information needed to compute the full system dynamics matrix. Tests and histories are arranged length-lexicographically, with ever increasing test and history lengths. The matrix has an infinite number of rows and columns, and like the system dynamics vector, it is a complete description of a dynamical system.

13.2.5 Sufficient Statistics

The system dynamics matrix inherently defines a notion of sufficient statistic, and suggests several possible learning algorithms and state update mechanisms. For example, even though the system dynamics matrix has an infinite number of columns and rows, if it has finite rank, there must be at least one finite set of linearly independent columns corresponding to a set of linearly independent tests. We call the tests associated with these linearly independent columns *core tests*. Similarly, there must be at least one set of linearly independent rows corresponding to linearly independent histories. We call the histories associated with these linearly independent rows *core histories*.

In fact, the rank of the system dynamics matrix has been shown to be finite for interesting cases, such as POMDPs, as explained in Section 13.2.10. Estimating and leveraging the rank of the systems dynamics matrix is a key component to many PSR learning algorithms, as we shall see.

13.2.6 State

We are now prepared to discuss the key idea of PSRs: PSRs represent state as a set of *predictions about core tests*, which represent the probabilities of possible future observations given possible future actions. Core tests are at the heart of PSRs, because by definition, every other column—i.e., any prediction about possible futures—can be computed as a weighted combination of these columns. Importantly, the weights that are independent of time and history, which means they may be estimated once and used for the lifetime of the agent.

To see how the predictions of a set of core tests can constitute state, consider a particular history h_t . Suppose that an agent knows which tests are core tests. We will call this set Q , and suppose furthermore that the agent has access to a vector containing their predictions from history h_t :

$$p(Q|h_t) = [p(q_1|h_t), p(q_2|h_t), \dots, p(q_n|h_t)].$$

Every other column c in the row corresponding to h_t can be computed as some weighted combination of the entries in $p(Q|h_t)$:

$$p(c|h_t) = m_c^\top p(Q|h_t).$$

Because columns correspond to possible futures, this agent can predict anything about the future that it needs to, assuming it has the appropriate weight vector m_c . The weight vector m_c does not depend on history, which will be critical to maintaining state, as we will see in a moment. Because an agent can predict anything it needs to as a linear combination of the entries in $p(Q|h_t)$, we say that the predictions of these core tests are a *linearly sufficient statistic* for the system.

We have thus satisfied the key definition of state outlined in the introduction: that $p(\text{future}|\text{state,past}) = p(\text{future}|\text{state})$. In this case, once we have the predictions of the core tests, we can compute any future of interest, and we may discard history.

13.2.7 State Update

We have shown how, given a set of predictions, we can compute probabilities of different futures. But how can we maintain those predictions in response to new actions and observations?

The probabilistic definition of tests helps us accomplish this. Assume that we are given a set of core tests \mathcal{Q} and their predictions $p(Q|h)$ at the current history h , and that we have taken a new action a , and received a new observation o . To update state, we need to compute $p(Q|hao)$, which is state from the new history hao (hao means “the sequence of actions and observations in h , followed by a new action a and observation o ”). We must compute $p(Q|hao)$ using only information in $p(Q|h)$. By definition, the updated prediction for a core test $q_i \in \mathcal{Q}$ is

$$p(q_i|hao) = \frac{p(aoq_i|h)}{p(ao|h)}.$$

Here, the notation aoq_i means “the test consisting of taking action a , observing o , and then taking the rest of the actions specified by q_i and seeing the corresponding observations in q_i .” Note that the quantities on the right-hand side are defined strictly in terms of predictions from history h —and because $p(Q|h)$ can be used to predict any future from h , we have solved the problem: to maintain state, we only need to compute the predictions of the *one step tests* (ao) and the *one-step extensions* (aoq_i) to the core tests as a function of $p(Q|h)$.

This formula is true for all PSRs, whether they use linearly sufficient statistics or nonlinearly sufficient statistics. In the case of linearly sufficient statistics, the state update takes on a particularly convenient form, as discussed next.

13.2.8 Linear PSRs

Linear PSRs are built around the idea of linearly sufficient statistics. In a linear PSR, for every test c , there is a weight vector $m_c \in \mathbb{R}^{|\mathcal{Q}|}$ independent of history h such that the prediction $p(c|h) = m_c^\top p(Q|h)$ for all h . This means that updating the prediction of a single core test $q_i \in \mathcal{Q}$ can be done efficiently in closed-form. From history h , after taking action a and seeing observation o :

$$p(q_i|hao) = \frac{p(aoq_i|h)}{p(ao|h)} = \frac{m_{aoq_i}^\top p(Q|h)}{m_{ao}^\top p(Q|h)}. \quad (13.1)$$

This equation shows how a single test can be recursively updated in an elegant, closed-form way. Previously, we said that given the predictions of a set of core tests for a certain history h , any other column in the same row could be computed as a weighted sum of $p(Q|h)$. Here, we see that in order to update state, only two predictions are needed: the prediction of the *one-step test* $p(ao|h)$, and the prediction of the *one-step extension* $p(aoq_i|h)$. Thus, the agent only needs to know the weight vectors m_{ao} , which are the weights for the one-step tests, and the m_{aoq_i} , which are the weights for the one-step extensions. We can combine the updates for all the core tests into a single update by defining the matrix M_{ao} , who's i 'th row is equal to $m_{aoq_i}^\top$. Updating all core tests simultaneously is now equivalent to a normalized matrix-vector multiply:

$$p(Q|hao) = \frac{M_{ao}p(Q|h)}{m_{ao}^\top p(Q|h)} \quad (13.2)$$

which allows us to recursively update state.

An agent does not need to learn a weight vector for every possible test in the system that it ever wishes to predict. If it has learned the weights for the one-step tests and the one-step extensions, these are sufficient to create a prediction for *any* arbitrary test. This is accomplished by rolling the model forward into the future. The prediction of an arbitrary test $t = a^1 o^1 \cdots a^n o^n$ can be computed as:

$$p(t|h) = m_{a^n o^n}^\top M_{a^{n-1} o^{n-1}} \cdots M_{a^1 o^1} p(Q|h). \quad (13.3)$$

This can be derived by considering the system dynamics matrix (Singh et al, 2004), or by considering the parameters of an equivalent POMDP (Littman et al, 2002).

13.2.9 Relating Linear PSRs to POMDPs

Many theoretical properties of linear PSRs have been established by examining the mappings between linear PSRs and POMDPs. Here, we will examine one relationship which provides some intuition for their general relationship: we show that both models have linear update and prediction equations, and that the parameters of a linear PSR are a similarity transformation of the equivalent POMDP's parameters.

In this section, we only consider finite POMDPs with n states. We use $O^{a,o}$ to represent an observation matrix, which is a diagonal $n \times n$ matrix. Entry $O_{i,i}^{a,o}$ represents the probability of observation o in state i when action a is taken. T^a is the $n \times n$ transition matrix with columns representing $T(s'|s,a)$.

Updating state in a POMDP is like updating state in a PSR. Given a belief state b_h and a new action a and observation o , the belief state b_{hao} is

$$b_{hao} = \frac{O^{a,o} T^a b_h}{\mathbf{1}^T O^{a,o} T^a b_h}$$

where $\mathbf{1}$ is the $n \times 1$ vector of all 1s. Note the strong similarity to the PSR state update in Eq. 13.2: the matrix product $O^{a,o}T^a$ can be computed once offline to create a single matrix, meaning the computational complexity of updating state in both representations is identical.

To compute $p(t|h)$ where $t = a^1 o^1 a^2 o^2 \cdots a^n o^n$, the POMDP must compute

$$p(t|h) = \mathbf{1}^T O^{a^n, o^n} T^{a^n} \cdots O^{a^1, o^1} T^{a^1} b_h = w_t^T b_h$$

where we can again precompute the vector-matrix products. Note the similarity to the PSR prediction equation in Eq. 13.3: both can make any prediction as a weighted combination of entries in their respective state vectors, with weights that do not depend on history.

Linear PSRs and POMDPs also have closely related parameters. To see this, we define the *outcome matrix* U for an n -dimensional POMDP and a given set of n core tests Q . An entry U_{ij} represents the probability that a particular test j will succeed if executed from state i . Note that U will have full rank if the set tests Q constitute a set of core tests. Given U , it has been shown (Littman et al, 2002) that:

$$\begin{aligned} M_{ao} &= U^{-1} T^a O^{a,o} U \\ m_{ao} &= U^{-1} T^a O^{a,o} \mathbf{1} \end{aligned}$$

In other words, the parameters of a linear PSR are a similarity transformation of the corresponding parameters of the equivalent POMDP. This fact turns out to be useful in translating POMDP planning algorithms to PSRs, as we shall see in Section 13.4.

13.2.10 Theoretical Results on Linear PSRs

There are a variety of theoretical results on the properties of Linear PSRs. Here, we survey a few of the most important.

Expressivity. It was shown early that every POMDP can be equivalently expressed by a PSR using a constructive proof translating from POMDPs directly to PSRs (Littman et al, 2002). This result was generalized by James (2005), who states that “Finite PSRs are able to model all finite POMDPs, HMMs, MDPs, MCs, history-window frameworks, diversity representations, interpretable OOMs and interpretable IO-OOMs.” In fact, PSRs are strictly more expressive than POMDPs. James (2005) showed that “there exist finite PSRs which cannot be modeled by any finite POMDP, Hidden Markov Model, MDP, Markov chain, history-window, diversity representation, interpretable OOM, or interpretable IO-OOM.”

Compactness. PSRs are just as compact as POMDPs. James (2005) states that “Every system that can be modeled by a finite POMDP (including all HMMs, MDPs, and MCs) can also be modeled by a finite PSR using number of core tests less than or equal to the number of nominal-states in the POMDP” (Thm. 4.2). In

addition, there are strong bounds on the length of the core tests needed to capture state. Wolfe (2009) showed that “For any system-dynamics matrix of rank n , there exists some set T of core tests such that no $t \in T$ has length greater than n .” (Thm. 2.4). Similar statements apply for core histories (Thm. 2.5).

Converting models between other frameworks and PSRs. While we have focused on defining PSRs via the systems dynamics vector and matrix, one can also derive PSR models from other models. An algorithm for converting POMDPs into equivalent PSRs was given by Littman et al (2002), and James presented an additional algorithm to convert an interpretable Observable Operator Model (OOM) to a PSR (James, 2005). Interestingly, there is no known method of recovering a POMDP from a PSR (perhaps because there are multiple POMDPs which map to the same PSR). If there were, then any learning algorithm for a PSR would become a learning algorithm for a POMDP.

13.3 Learning a PSR Model

Numerous algorithms have been proposed to learn PSRs, but they all address two key problems: the *discovery problem* and the *learning problem* (James and Singh, 2004). The discovery problem is defined as the problem of finding a set of core tests, and is essentially the problem of discovering a state representation. The learning problem is defined as the problem of finding the parameters of the model needed to update state, and is essentially the problem of learning the dynamical aspect of the system. In the case of linear PSRs, this is the m_{qi} weight vectors for all of the one-step tests and the one-step extensions.

13.3.1 The Discovery Problem

The idea of linear sufficiency suggests procedures for discovering sufficient statistics: a set of core tests corresponds to a set of linearly independent columns of the system dynamics matrix, and so techniques from linear algebra can be brought to bear on empirical estimates of portions of the system dynamics matrix. Existing discovery algorithms search for linearly independent columns, which is a challenging task because the columns of the matrix are estimated from data, and noisy columns are often linearly independent (Jaeger, 2004). Thus, the *numerical* rank of the matrix must be estimated using a statistical test based on the singular values of the matrix. The entire procedure typically relies on repeated singular value decompositions of the matrix, which is costly. For example, James and Singh (2004) learn a “history-test matrix,” which is the predecessor to the systems dynamics matrix. Their algorithm repeatedly estimates larger and larger portions of the matrix,

until a stopping criterion is reached. An alternative strategy is avoid a search for minimally sufficient statistics entirely, and use an overcomplete basis by using a very large number of randomly sampled tests (Wingate and Singh, 2008); no theoretical properties of such an approach are known.

13.3.2 The Learning Problem

Once the core tests have been found, the update parameters must be learned. Singh et al (2003) presented the first algorithm for the learning problem, which assumes that the core tests are given and uses a gradient algorithm to compute weights. The more common approach is with regression and sample statistics (James and Singh, 2004): once a set of core tests is given, the update parameters can be solved by regressing the appropriate entries of the estimated system dynamics matrix.

Some authors combine both problems into a single algorithm. For example, Wiewiora (2005) presents a method for learning regular form PSRs with an iterative extend-and-regress method, while McCracken and Bowling (2006) propose an online discovery and learning algorithm based on gradient descent.

Not every set of matrices M_{ao} define a valid linear PSR. If an estimation algorithm returns invalid parameters, PSRs can suffer from underflow (predicting negative probabilities) or overflow (predicting probabilities greater than unity). Precise constraints on PSR parameters were identified by Wolfe (2010), who used the constraints to improve estimates of the parameters, and demonstrated that predictions generated from the resulting model did not suffer from underflow or overflow.

13.3.3 Estimating the System Dynamics Matrix

Many learning and discovery algorithms involve estimating the system dynamics matrix, typically using sample statistics. In systems with a reset action, the agent may actively reset to the empty history in order to repeatedly sample entries (James and Singh, 2004). In systems without a reset, most researchers use the suffix-history algorithm (Wolfe et al, 2005) to generate samples: given a trajectory of the system, we slice the trajectory into all possible histories and futures. Active exploration is also possible, as proposed by Bowling et al (2006).

13.4 Planning with PSRs

There has been comparatively little work on planning in PSRs, partly because it is generally believed among PSR researchers that any POMDP planning algorithm can be directly applied to PSRs (although there is no formal proof of this). This is partly

a consequence of several foundational proofs proving that value functions in PSRs and POMDPs have similar properties, and also from the general observation that “For any dynamical system, the POMDP-to-PSR mapping preserves the pairwise distance between points in the space of reachable beliefs, within a constant factor” (Izadi and Precup, 2008). This is a consequence of the linear relationship between PSRs and their equivalent POMDPs discussed in Section 13.2.9, and means that researchers have had little difficulty in translating POMDP algorithms to PSRs.

James et al (2004) established the first fundamental results. They showed that a policy tree can be constructed for a PSR, like a POMDP, and that the value function for PSRs is a piecewise linear and convex function over prediction vectors, just as the value function in POMDPs is a piecewise linear and convex function over belief state vectors. This motivated PSR versions of Incremental Pruning and Q-learning, which empirically performed comparably to their POMDP counterparts.

Later work by James et al (2006) demonstrated that a randomized point-based planning method similar to PERSEUS could be directly applied to PSRs; again, POMDP and PSR performance was similar. A similar result was obtained by Izadi and Precup (2008), who contributed another PSR version of point-based value iteration, claiming that their method “outperforms the original PBVI when the PSR representation introduces some degree of compression compared to the corresponding POMDP. Otherwise, the results are comparable.” Additional work has shown that planning using memory PSRs can further improve performance over vanilla PSRs (James et al, 2004, 2006).

Policy gradients have also been applied to PSRs, with mixed results. Aberdeen et al (2007) applied the Natural Actor-Critic (Peters et al, 2005) policy gradient algorithm to PSRs in several benchmark PSR/POMDP domains, concluding that it appears to work equally well under both models. Wingate (2008) also applied NAC (as well as Least-Squares Policy Iteration (Lagoudakis and Parr, 2003)) to the Exponential Family PSR (EFPSR), concluding that for their particular problem the state representation was sufficiently accurate to support NAC but not LSPI. One unusual combination of planning in partially observable domains and PSRs was given by Bouali and Chaib-draa (2009), who observed that just as PSRs can replace POMDPs as a dynamics model, PSRs can replace Finite State Controllers (FSC) as a policy model. They derived a policy gradient method for such a PSR-FSC and demonstrated that it typically outperformed the more traditional FSC.

To summarize, planning in PSRs does not seem any harder or easier than planning in POMDPs. There have not yet been any outstanding successes or failures; in general, plans achieved with PSRs are competitive with those obtained with POMDPs. And while no particularly novel method has been developed which leverages the unique representations of PSRs, many authors have concluded that planning with a PSR representation will generally be slightly better than or equal to planning with an equivalent POMDP representation.

13.5 Extensions of PSRs

The basic PSR model has been extended in numerous ways, mostly in efforts to scale them to cope with larger domains. This is typically done by leveraging some sort of structure in the domain, although it can also be achieved by adopting a more powerful (i.e., nonlinear) state update method.

Memory PSRs. James et al (2005b) showed that memory and predictions can be combined to yield smaller models than can be obtained solely with predictions. Their Memory-PSR (or mPSR) “remembers” an observation (typically the most recent one), and builds a separate PSR for the distribution of the future conditioned on each unique memory. This can result in compact, factored models if the memories form landmarks that allow the domain to be decomposed. James and Singh (2005a) then showed that effective planning is possible with the resulting model.

Hierarchical PSRs. Wolfe (2009) considers temporal abstraction with hPSRs: instead of interleaving actions and observations, he interleaves options (Precup et al, 1998) and observations. These options capture temporal structure in the domain by compressing regularities in action-observation sequences, which allows scaling to larger domains. For example, in a maze domain, long hallways or large rooms may have a simple compressed representation. Wolfe analyzes the resulting “option-level” dynamical system as a compression of the system dynamics matrix which sums out rows and columns. Counter to intuition, the resulting matrix can have a larger rank than the original system (although he provides conditions where the rank of the option system is no more than the rank of the original system [Thm 4.5]).

Factored PSRs. While hPSRs leverage temporal structure in a domain, Factored PSRs (Wolfe, 2009) leverage structure among the observation dimensions of a dynamical system with vector-valued observations by exploiting conditional independence between subsets of observation dimensions. This can be important for domains with many mostly independent observations (the motivating example is a traffic prediction problem where observations consist of information about hundreds of cars on a freeway). An important feature of Factored PSRs is that the representation can remain factored even when making long-term predictions about the future state of the system. This is in contrast to, say, a DBN, where predictions about latent states or observations many time steps in the future will often be dependent upon all of the current latent state variables. The intuition behind this is explained by Wolfe:

“Consider the i -th latent state variable [of a DBN] at the current time step. It will affect some subset of latent state variables at the next time step; that subset will affect another subset of variables at the following time step; and so on. Generally speaking, the subset of variables that can be traced back to the i -th latent state variable at the current time step will continue to grow as one looks further forward in time. Consequently, in general, the belief state of a DBN—which tracks the joint distribution of the latent state variables—is not factored, and has size exponential in the number of latent state variables. In contrast, the factored PSR does not reference latent state, but only actions and observations.”

Multi-Mode PSRs. The Multi-mode PSR model (Wolfe et al, 2008, 2010) is designed to capture uncontrolled systems that switch between several modes of operation. The MMPSR makes specialized predictions conditioned upon the current mode, which can simplify the overall model. The MMPSR was inspired by the problem of predicting cars’ movements on a highway, where cars typically operate in discrete modes of operation like making a lane change, or passing another car. These modes are like options, but modes are not latent, unobservable quantities (as opposed to, say, upper levels in a hierarchical HMM). Wolfe defines a *recognizability condition* to identify modes, which can be a function of both past and future observations, but emphasizes that this does not limit the class of systems that can be modeled; it merely impacts the set of modes available to help model the system.

Relational PSRs. Wingate et al (2007) showed how PSRs can be used to model relational domains by grounding all predicates in predictions of the future. One contribution of this work was the introduction of new kinds of tests: set tests (which make predictions about sequences of actions and *sets* of observations) and indexical tests (which make predictions about sequences of actions and observations where an action/observation at time t is required to have the same—but unspecified—value as an action/observation at a later time $t + k$, thus allowing a sort of “variable” in the test). These tests permit a great deal of flexibility, but still retain all of the linearity properties of linear PSRs, making them expressive and tractable.

Continuous PSRs. There has been less work on the systems with continuous actions and observations. Wingate and Singh (2007b) present a generalization of PSRs where tests predict the *density* of a sequence of continuous actions and observations. This creates a problem: PSRs with discrete observations derive the notion of predictive state as a sufficient statistic via the rank of the system dynamics matrix, but with continuous observations and actions, such a matrix and its rank no longer exist. The authors define an analogous construct for the continuous case, called the system dynamics distributions, and use information theoretic notions to define a sufficient statistic and thus state. They use kernel density estimation to learn a model, and information gradients to directly optimize the state representation.

Other extensions. Tanner et al (2007) presented a method to learn high-level abstract features from low-level state representations. On the state update side, Rudary and Singh (2004) showed that linear models can be compacted when nonlinearly sufficient statistics are allowed.

13.6 Other Models with Predictively Defined State

There are other models of dynamical systems which capture state through the use of predictions about the future.

13.6.1 *Observable Operator Models*

Observable Operator Models (OOMs) were introduced and studied by Jaeger (2000). Like PSRs, there are several variants on the same basic theme, making it more of a framework than a single model. Within the family of OOMs are models which are designed to deal with different versions of dynamical systems: the basic OOM models uncontrolled dynamical systems, while the IO-OOM models controlled dynamical systems. OOMs have several similarities to PSRs. For example, there are analogous constructs to core tests (“characteristic events”), core histories (“indicative events”) and the system dynamics matrix. State in an OOM is represented as a vector of predictions about the future, but the predictions do not correspond to a single test. Instead, each entry in the state vector is the prediction of some set of tests of the same length k . There are constraints on these sets: they must be disjoint, but their union must cover all tests of length k .

A significant restriction on IO-OOMs is that the action sequence used in tests must be the same for all tests, which is needed to satisfy some assumptions about the state vector. The assumption is severe enough that James (2005) gives an example of why it results in systems which the IO-OOM cannot model, but which PSRs can. There are also variants of OOMs which do not use predictions as part of their state representation (called “uninterpretable OOMs”) but there are no learning algorithms for these models (James, 2005). We refer the reader to the technical report by Jaeger (2004) for a detailed comparison of PSRs and OOMs.

13.6.2 *The Predictive Linear-Gaussian Model*

The “Predictive Linear-Gaussian” model (PLG)(Rudary et al, 2005) is a generalization of PSRs to uncontrolled systems with linear dynamics and simple scalar observations. The PLG defines state as the parameters of a Gaussian distribution over a window of the short-term future. This is another generalization of PSR style representations: state is defined as a mean and a covariance matrix over the future, both of which can be interpreted as the expectation of features of the future.

The PLG is another example of a predictively defined model which is representationally equivalent to its classical counterpart. The PLG is formally equivalent in modeling capacity to a linear dynamical system: at any time t , an LDS or a PLG can be used to predict a distribution over future observations, and these distributions are identical. Furthermore, the PLG is just as compact as the corresponding LDS: an n -dimensional LDS can be captured by an n -dimensional PLG, and state in the PLG can be maintained with equations that are equal in computational complexity to the standard Kalman Filter update equations (Kalman, 1960). The basic PLG has been extended in many ways including vector-valued observations (Rudary and Singh, 2008), and linear controls (Rudary and Singh, 2006). For the case of a Linear Quadratic Regulator, it has been further shown that the optimal policy of the PLG is a linear function of state (just like the classic LDS), and that it can be found

with equations analogous to the Riccati equations. Like PSRs, the PLG has learning algorithms that are based on sample statistics and regressions. An important learnability result was obtained by Rudary et al (2005), who showed a statistically consistent parameter estimation algorithm, which is an important contrast to typical LDS learning based on methods such as EM (Ghahramani and Hinton, 1996).

Nonlinear dynamics have been considered in two different ways. By playing the kernel trick, linear dynamics can be represented in a nonlinear feature space. This results in the Kernel PLG (Wingate and Singh, 2006a). A more robust and learnable method seems to be assuming that dynamics are piecewise linear, resulting in the “Mixtures of PLGs” model (Wingate and Singh, 2006b). In many ways, these extensions parallel the development of the Kalman filter: for example, state in the KPLG can be updated with an efficient approximate inference algorithm based on sigma-point approximations (yielding an algorithm related to the unscented Kalman filter (Wan and van der Merwe, 2000)).

13.6.3 Temporal-Difference Networks

The Temporal-Difference Network model of Sutton and Tanner (2005) is an important generalization of PSRs. In a TD-Net, state is represented as a set of predictions about the future, like a PSR. However, these predictions are explicitly allowed to depend on each other in a compositional, recursive way. This suggests that temporal difference algorithms could be used to learn the predictions, as opposed to the Monte-Carlo methods used by PSRs, and it is these algorithms which form the basis of the model. The recursive nature of the tests and the use of temporal difference methods in learning naturally generalizes to include multi-step backups of learning by introducing eligibility traces, to form $\text{TD}(\lambda)$ -Nets (Tanner and Sutton, 2005a).

Although TD-Nets are theoretically attractive, they have not enjoyed the same rigorous analysis which PSRs have. Little is known about their representational capacity or the optimality of their state update mechanism. For example, published work on TD-Nets uses a general nonlinear state update mechanism related to a single-layer neural network, although this is not a fundamental component of the model. Other state updates could be used, and it is not clear how the state update relates to, say, the statistically optimal update dictated by Bayes law. PSRs, in contrast, explicitly begin with Bayes law as the foundation of their state update mechanism.

Empirically, TD-Nets have enjoyed about the same level of successes and failures as PSRs, with applications of the model being limited to rather small domains. While there has been less work done on TD-Nets in general, the development of learning algorithms for TD-Nets and PSRs have in some ways paralleled each other. For example, Tanner and Sutton (2005b) proposed to include some history in the state representation to aid learning in a manner that is reminiscent of the memory PSRs proposed by James and Singh (2005a), with improved learning results.

13.6.4 Diversity Automaton

The diversity automaton of Rivest and Schapire (1987) is a model based on predictions about the future, although with some severe restrictions. Like the PSR model, diversity models represent state as a vector of predictions about the future. However, these predictions are not as flexible as the usual tests used by PSRs, but rather are limited to be like the e-tests used by Rudary and Singh (2004). Each test t_i is the probability that a certain observation will occur in n_i steps, given a string of n_i actions but *not* given any observations between time $t + 1$ and $t + n_i$. Each of these tests corresponds to an equivalence class over the distribution of future observations.

Rivest and Schapire (1987) showed tight bounds on the number of tests needed by a diversity model relative to the number of states a minimal POMDP would need. Diversity models can either compress or inflate a system: in the best case, a logarithmic number of tests are needed, but in the worst case, an exponential number are needed. This contrasts with PSRs, where only n tests to model any domain modeled by an n -state POMDP. Diversity models are also limited to systems with deterministic transitions and deterministic observations. This is due to the model's state update mechanism and the need to restrict the model to a finite number of tests by restricting it to a finite number of equivalence classes of future distributions.

13.6.5 The Exponential Family PSR

The exponential family PSR, or EFPSR, is a general model unifying many other models with predictively defined state (Wingate and Singh, 2007a). It was motivated by the observation that these models track the sufficient statistics of an exponential family distribution over the short-term future. For example, the PLG uses the parameters of a Gaussian, while the PSR uses the parameters of a multinomial. An exponential family distribution is any distribution which can be written as $p(\text{future}|\text{history}) \propto \exp\{w_t^T \phi(\text{future})\}$; Suitable choices of $\phi(\text{future})$ and an update mechanism for w_t recover existing models.

This leads to the idea of placing a general exponential family distribution over the short-term future observations, parameterized with features ϕ of the future. This generalization has been used to predict and analyze new models, including the Linear-Linear EFPSR (Wingate and Singh, 2008) which is designed to model domains with large numbers of features, and has also resulted in an information-form of the PLG (Wingate, 2008). This results in strong connections between graphical models, maximum entropy modeling, and PSRs.

From a purist's point of view, it is questionable whether the EFPSR is really a PSR: while the representation is defined in terms of the short-term distribution over the future, the parameters of the model are less directly grounded in data: they are not verifiable in the same way that an expected value of a future event is. The model therefore sits somewhere in between latent state space models and PSRs.

13.6.6 Transformed PSRs

Transformed PSRs (Rosencrantz et al, 2004) also learn a model by estimating the system dynamics matrix. However, instead of searching for linearly independent columns (i.e., core tests), they perform a singular value decomposition on the system dynamics matrix to recover a low-dimensional representation of state, which can be interpreted as a linear combination of core tests. The resulting state is a rotated PSR, but elements of the state vector cannot be interpreted as predictions of tests. This work was subsequently generalized Boots et al (2010) with an alternative learning algorithm and the ability to deal with continuous observations. The overall result is one of the first learning algorithms that is statistically consistent and which empirically works well enough to support planning in complex domains.

13.7 Conclusion

Models with predictively defined representations of state are still relatively young, but some theoretical and empirical results have emerged. What broad conclusions can we draw from these results? What might their future hold?

Results on compactness, expressivity and learnability all suggest that the idea is viable. We have seen, for example, that there are strong connections between PSRs and POMDPs: PSRs are at least as compact as POMDPs and have comparable computational complexity, and are strictly more expressive. Like other models, PSRs can leverage structure in domains, as demonstrated by extensions to factored, hierarchical, switching, kernelized, and continuous domains. Of course, the real promise of the idea is learnability, and we have seen a variety of functional learning algorithms—along with some tantalizing first glimpses of optimal learnability in the statistical consistency results of PLGs and TPSRs.

Predictive representations are also surprisingly flexible. Different authors have captured state by using probabilities of specific tests (the PSR model), densities of specific tests (the Continuous PSR model), expectations of features of the short-term future (the PLG family of models), the natural parameters of a distribution over a window of short-term future observations (the EFPSR), and wide variety of tests (including set tests, indexical tests, and option tests). It is likely that there are many other possibilities which are as yet unexplored.

Just as important are an absence of serious negative results. As researchers have explored this space, there has always been the question: is there anything a model with a predictively defined representation of state can *not* do? Would this representation be a limitation? So far, the answer is no: there is no known representational, computational, theoretical or empirical limitation that has resulted from adopting a predictively defined representation of state, and indeed, there are still good reasons to believe they have special properties which uniquely suit them to a variety of tasks.

Has their full depth been plumbed? It seems unlikely. But perhaps one of their most significant contributions has already been made: to broaden our thinking by challenging us to reconsider what it means to learn and represent state.

References

- Aberdeen, D., Buffet, O., Thomas, O.: Policy-gradients for psrs and pomdps. In: International Workshop on Artificial Intelligence and Statistics, AISTAT (2007)
- Astrom, K.J.: Optimal control of Markov decision processes with the incomplete state estimation. *Journal of Computer and System Sciences* 10, 174–205 (1965)
- Boots, B., Siddiqi, S., Gordon, G.: Closing the learning-planning loop with predictive state representations. In: Proceedings of Robotics: Science and Systems VI, RSS (2010)
- Boualiyas, A., Chaib-draa, B.: Predictive representations for policy gradient in pomdps. In: International Conference on Machine Learning, ICML (2009)
- Bowling, M., McCracken, P., James, M., Neufeld, J., Wilkinson, D.: Learning predictive state representations using non-blind policies. In: International Conference on Machine Learning (ICML), pp. 129–136 (2006)
- Ghahramani, Z., Hinton, G.E.: Parameter estimation for linear dynamical systems. Tech. Rep. CRG-TR-96-2, Dept. of Computer Science, U. of Toronto (1996)
- Izadi, M., Precup, D.: Model minimization by linear psr. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 1749–1750 (2005)
- Izadi, M.T., Precup, D.: Point-Based Planning for Predictive State Representations. In: Bergler, S. (ed.) Canadian AI. LNCS (LNAI), vol. 5032, pp. 126–137. Springer, Heidelberg (2008)
- Jaeger, H.: Observable operator processes and conditioned continuation representations. *Neural Computation* 12(6), 1371–1398 (2000)
- Jaeger, H.: Discrete-time, discrete-valued observable operator models: A tutorial. Tech. rep., International University Bremen (2004)
- James, M., Singh, S., Littman, M.: Planning with predictive state representations. In: International Conference on Machine Learning and Applications (ICMLA), pp. 304–311 (2004)
- James, M., Wessling, T., Vlassis, N.: Improving approximate value iteration using memories and predictive state representations. In: Proceedings of AAAI (2006)
- James, M.R.: Using predictions for planning and modeling in stochastic environments. PhD thesis, University of Michigan (2005)
- James, M.R., Singh, S.: Learning and discovery of predictive state representations in dynamical systems with reset. In: International Conference on Machine Learning (ICML), pp. 417–424 (2004)
- James, M.R., Singh, S.: Planning in models that combine memory with predictive representations of state. In: National Conference on Artificial Intelligence (AAAI), pp. 987–992 (2005a)
- James, M.R., Wolfe, B., Singh, S.: Combining memory and landmarks with predictive state representations. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 734–739 (2005b)
- Kalman, R.E.: A new approach to linear filtering and prediction problem. *Transactions of the ASME—Journal of Basic Engineering* 82(Series D), 35–45 (1960)
- Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. *Journal of Machine Learning Research (JMLR)* 4, 1107–1149 (2003)

- Littman, M.L., Sutton, R.S., Singh, S.: Predictive representations of state. In: Neural Information Processing Systems (NIPS), pp. 1555–1561 (2002)
- McCracken, P., Bowling, M.: Online discovery and learning of predictive state representations. In: Neural Information Processings Systems (NIPS), pp. 875–882 (2006)
- Nikovski, D.: State-aggregation algorithms for learning probabilistic models for robot control. PhD thesis, Carnegie Mellon University (2002)
- Peters, J., Vijayakumar, S., Schaal, S.: Natural actor-critic. In: European Conference on Machine Learning (ECML), pp. 280–291 (2005)
- Precup, D., Sutton, R.S., Singh, S.: Theoretical results on reinforcement learning with temporally abstract options. In: European Conference on Machine Learning (ECML), pp. 382–393 (1998)
- Rafols, E.J., Ring, M.B., Sutton, R.S., Tanner, B.: Using predictive representations to improve generalization in reinforcement learning. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 835–840 (2005)
- Rivest, R.L., Schapire, R.E.: Diversity-based inference of finite automata. In: IEEE Symposium on the Foundations of Computer Science, pp. 78–87 (1987)
- Rosencrantz, M., Gordon, G., Thrun, S.: Learning low dimensional predictive representations. In: International Conference on Machine Learning (ICML), pp. 695–702 (2004)
- Rudary, M., Singh, S.: Predictive linear-Gaussian models of stochastic dynamical systems with vector-value actions and observations. In: Proceedings of the Tenth International Symposium on Artificial Intelligence and Mathematics, ISAIM (2008)
- Rudary, M.R., Singh, S.: A nonlinear predictive state representation. Neural Information Processing Systems (NIPS), 855–862 (2004)
- Rudary, M.R., Singh, S.: Predictive linear-Gaussian models of controlled stochastic dynamical systems. In: International Conference on Machine Learning (ICML), pp. 777–784 (2006)
- Rudary, M.R., Singh, S., Wingate, D.: Predictive linear-Gaussian models of stochastic dynamical systems. In: Uncertainty in Artificial Intelligence, pp. 501–508 (2005)
- Shatkay, H., Kaelbling, L.P.: Learning geometrically-constrained hidden Markov models for robot navigation: Bridging the geometrical-topological gap. Journal of AI Research (JAIR), 167–207 (2002)
- Singh, S., Littman, M., Jong, N., Pardoe, D., Stone, P.: Learning predictive state representations. In: International Conference on Machine Learning (ICML), pp. 712–719 (2003)
- Singh, S., James, M.R., Rudary, M.R.: Predictive state representations: A new theory for modeling dynamical systems. In: Uncertainty in Artificial Intelligence (UAI), pp. 512–519 (2004)
- Sutton, R.S., Tanner, B.: Temporal-difference networks. In: Neural Information Processing Systems (NIPS), pp. 1377–1384 (2005)
- Tanner, B., Sutton, R.: $Td(\lambda)$ networks: Temporal difference networks with eligibility traces. In: International Conference on Machine Learning (ICML), pp. 888–895 (2005a)
- Tanner, B., Sutton, R.: b) Temporal difference networks with history. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 865–870 (2005)
- Tanner, B., Bulitko, V., Koop, A., Paduraru, C.: Grounding abstractions in predictive state representations. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 1077–1082 (2007)
- Wan, E.A., van der Merwe, R.: The unscented Kalman filter for nonlinear estimation. In: Proceedings of Symposium 2000 on Adaptive Systems for Signal Processing, Communication and Control (2000)

- Wiewiora, E.: Learning predictive representations from a history. In: International Conference on Machine Learning (ICML), pp. 964–971 (2005)
- Wingate, D.: Exponential family predictive representations of state. PhD thesis, University of Michigan (2008)
- Wingate, D., Singh, S.: Kernel predictive linear Gaussian models for nonlinear stochastic dynamical systems. In: International Conference on Machine Learning (ICML), pp. 1017–1024 (2006a)
- Wingate, D., Singh, S.: Mixtures of predictive linear Gaussian models for nonlinear stochastic dynamical systems. In: National Conference on Artificial Intelligence (AAAI) (2006b)
- Wingate, D., Singh, S.: Exponential family predictive representations of state. In: Neural Information Processing Systems, NIPS (2007a) (to appear)
- Wingate, D., Singh, S.: On discovery and learning of models with predictive representations of state for agents with continuous actions and observations. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 1128–1135 (2007b)
- Wingate, D., Singh, S.: Efficiently learning linear-linear exponential family predictive representations of state. In: International Conference on Machine Learning, ICML (2008)
- Wingate, D., Soni, V., Wolfe, B., Singh, S.: Relational knowledge with predictive representations of state. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 2035–2040 (2007)
- Wolfe, B.: Modeling dynamical systems with structured predictive state representations. PhD thesis, University of Michigan (2009)
- Wolfe, B.: Valid parameters for predictive state representations. In: Eleventh International Symposium on Artificial Intelligence and Mathematics (ISAIM) (2010)
- Wolfe, B., James, M.R., Singh, S.: Learning predictive state representations in dynamical systems without reset. In: International Conference on Machine Learning, pp. 980–987 (2005)
- Wolfe, B., James, M., Singh, S.: Approximate predictive state representations. In: Proceedings of the 2008 International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2008)
- Wolfe, B., James, M., Singh, S.: Modeling multiple-mode systems with predictive state representations. In: Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems (2010)

Chapter 14

Game Theory and Multi-agent Reinforcement Learning

Ann Nowé, Peter Vrancx, and Yann-Michaël De Hauwere

Abstract. Reinforcement Learning was originally developed for Markov Decision Processes (MDPs). It allows a single agent to learn a policy that maximizes a possibly delayed reward signal in a stochastic stationary environment. It guarantees convergence to the optimal policy, provided that the agent can sufficiently experiment and the environment in which it is operating is Markovian. However, when multiple agents apply reinforcement learning in a shared environment, this might be beyond the MDP model. In such systems, the optimal policy of an agent depends not only on the environment, but on the policies of the other agents as well. These situations arise naturally in a variety of domains, such as: robotics, telecommunications, economics, distributed control, auctions, traffic light control, etc. In these domains multi-agent learning is used, either because of the complexity of the domain or because control is inherently decentralized. In such systems it is important that agents are capable of discovering good solutions to the problem at hand either by coordinating with other learners or by competing with them. This chapter focuses on the application reinforcement learning techniques in multi-agent systems. We describe a basic learning framework based on the economic research into game theory, and illustrate the additional complexity that arises in such systems. We also described a representative selection of algorithms for the different areas of multi-agent reinforcement learning research.

14.1 Introduction

The reinforcement learning techniques studied throughout this book enable a single agent to learn optimal behavior through trial-and-error interactions with its environment. Various RL techniques have been developed which allow an agent to optimize

Ann Nowé · Peter Vrancx · Yann-Michaël De Hauwere
Vrije Universiteit Brussel
e-mail: {anowe, pvrancx, ydehauwe}@vub.ac.be

its behavior in a wide range of circumstances. However, when multiple learners simultaneously apply reinforcement learning in a shared environment, the traditional approaches often fail.

In the multi-agent setting, the assumptions that are needed to guarantee convergence are often violated. Even in the most basic case where agents share a stationary environment and need to learn a strategy for a single state, many new complexities arise. When agent objectives are aligned and all agents try to maximize the same reward signal, coordination is still required to reach the global optimum. When agents have opposing goals, a clear optimal solution may no longer exist. In this case, an equilibrium between agent strategies is usually searched for. In such an equilibrium, no agent can improve its payoff when the other agents keep their actions fixed.

When, in addition to multiple agents, we assume a dynamic environment which requires multiple sequential decisions, the problem becomes even more complex. Now agents do not only have to coordinate, they also have to take into account the current state of their environment. This problem is further complicated by the fact that agents typically have only limited information about the system. In general, they may not be able to observe actions or rewards of other agents, even though these actions have a direct impact on their own rewards and their environment. In the most challenging case, an agent may not even be aware of the presence of other agents, making the environment seem non-stationary. In other cases, the agents have access to all this information, but learning in a fully joint state-action space is in general impractical, both due to the computational complexity and in terms of the coordination required between the agents. In order to develop a successful multi-agent approach, all these issues need to be addressed. Figure 14.1 depicts a standard model of Multi-Agent Reinforcement Learning.

Despite the added learning complexity, a real need for multi-agent systems exists. Often systems are inherently decentralized, and a central, single agent learning approach is not feasible. This situation may arise because data or control is physically distributed, because multiple, possibly conflicting, objectives should be met, or simply because a single centralized controller requires too many resources. Examples of such systems are multi-robot set-ups, decentralized network routing, distributed load-balancing, electronic auctions, traffic control and many others.

The need for adaptive multi-agent systems, combined with the complexities of dealing with interacting learners has led to the development of a multi-agent reinforcement learning field, which is built on two basic pillars: the reinforcement learning research performed within AI, and the interdisciplinary work on game theory. While early game theory focused on purely competitive games, it has since developed into a general framework for analyzing strategic interactions. It has attracted interest from fields as diverse as psychology, economics and biology. With the advent of multi-agent systems, it has also gained importance within the AI community and computer science in general. In this chapter we discuss how game theory provides both a means to describe the problem setting for multi-agent learning and the tools to analyze the outcome of learning.

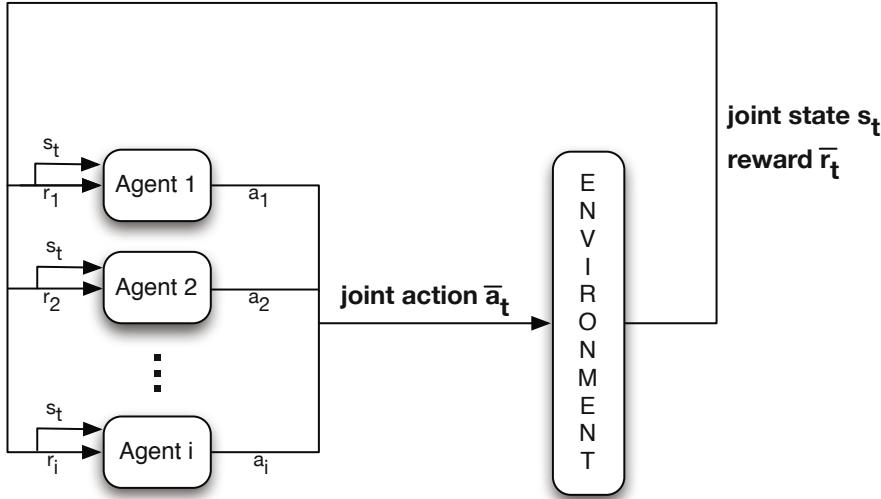


Fig. 14.1 Multiple agents acting in the same environment

The multi-agent systems considered in this chapter are characterized by strategic interactions between the agents. By this we mean that the agents are autonomous entities, who have individual goals and independent decision making capabilities, but who also are influenced by each other's decisions. We distinguish this setting from the approaches that can be regarded as distributed or parallel reinforcement learning. In such systems multiple learners collaboratively learn a single objective. This includes systems where multiple agents update the policy in parallel (Mariano and Morales, 2001), swarm based techniques (Dorigo and Stützle, 2004) and approaches dividing the learning state space among agents (Steenhaut et al, 1997). Many of these systems can be treated as advanced exploration techniques for standard reinforcement learning and are still covered by the single agent theoretical frameworks, such as the framework described in (Tsitsiklis, 1994). The convergence of the algorithms remain valid as long as outdated information is eventually discarded. For example, it allows to use outdated Q-values in the max-operator in the right hand side of standard Q-learning update rule (described in Chapter 1). This is particularly interesting when the Q-values are belonging to different agents each exploring their own part of the environment and only now and then exchange their Q-values. The systems covered by this chapter, however, go beyond the standard single agent theory, and as such require a different framework.

An overview of multi-agent research based on strategic interactions between agents is given in Table 14.1. The techniques listed are categorized based on their

applicability and kind of information they use while learning in a multi-agent system. We distinguish between techniques for stateless games, which focus on dealing with multi-agent interactions while assuming that the environment is stationary, and Markov game techniques, which deal with both multi-agent interactions and a dynamic environment. Furthermore, we also show the information used by the agents for learning. Independent learners learn based only on their own reward observation, while joint action learners also use observations of actions and possibly rewards of the other agents.

Table 14.1 Overview of current MARL approaches. Algorithms are classified by their applicability (common interest or general Markov games) and their information requirement (scalar feedback or joint-action information).

		Game setting		
		Stateless Games	Team Markov Games	General Markov Games
Information Requirement	Independent Learners	Stateless Q-learning Learning Automata IGA FMQ Commitment Sequences Lenient Q-learners	Policy Search Policy Gradient	MG-ILA (WoLF-)PG Learning of Coordination Independent RL CQ-learning
	Joint Action Learners		Distributed-Q Sparse Tabular Q Utile Coordination	Nash-Q Friend-or-Foe Q Asymmetric Q Joint Action Learning Correlated-Q

In the following section we will describe the repeated games framework. This setting introduces many of the complexities that arise from interactions between learning agents. However, the repeated game setting only considers static, stateless environments, where the learning challenges stem only from the interactions with other agents. In Section 14.3 we introduce Markov Games. This framework generalizes the Markov Decision Process (MDP) setting usually employed for single agent RL. It considers both interactions between agents and a dynamic environment. We explain both value iteration and policy iteration approaches for solving these Markov games. Section 14.4 describes the current state of the art in multi-agent research, which takes the middle ground between independent learning techniques and Markov game techniques operating in the full joint-state joint-action space. Finally in Section 14.5, we shortly describe other interesting background material.

14.2 Repeated Games

14.2.1 Game Theory

The central idea of game theory is to model strategic interactions as a game between a set of players. A game is a mathematical object, which describes the consequences of interactions between player strategies in terms of individual payoffs. Different representations for a game are possible. For example, traditional AI research often focusses on the *extensive form* games, which were used as a representation of situations where players take turns to perform an action. This representation is used, for instance, with the classical minimax algorithm (Russell and Norvig, 2003). In this chapter, however, we will focus on the so called *normal form* games, in which game players simultaneously select an individual action to perform. This setting is often used as a testbed for multi-agent learning approaches. Below we review basic game theoretic terminology and define some common solution concepts in games.

14.2.1.1 Normal Form Games

Definition 14.1. A normal form game is a tuple $(n, A_1, \dots, n, R_1, \dots, n)$, where

- $1, \dots, n$ is a collection of participants in the game, called players;
- A_k is the individual (finite) set of actions available to player k ;
- $R_k : A_1 \times \dots \times A_n \rightarrow \mathbb{R}$ is the individual reward function of player k , specifying the expected payoff he receives for a play $\mathbf{a} \in A_1 \times \dots \times A_n$.

A game is played by allowing each player k to independently select an individual action a from its private action set A_k . The combination of actions of all players constitute a *joint action* or *action profile* \mathbf{a} from the joint action set $\mathbb{A} = A_1 \times \dots \times A_n$. For each joint action $\mathbf{a} \in \mathbb{A}$, $R_k(\mathbf{a})$ denotes agent k 's expected payoff.

Normal form games are represented by their *payoff matrix*. Some typical 2-player games are given in Table 14.2. In this case the action selected by player 1 refers to a row in the matrix, while that of player 2 determines the column. The corresponding entry in the matrix then gives the payoffs player 1 and player 2 receive for the play. Players 1 and 2 are also referred to as the row and the column player, respectively. Using more dimensional matrices n -player games can be represented where each entry in the matrix contains the payoff for each of the agents for the corresponding combination of actions.

A *strategy* $\sigma_k : A_k \rightarrow [0, 1]$ is an element of $\mu(A_k)$, the set of probability distributions over the action set A_k of player k . A strategy is called pure if $\sigma_k(a) = 1$ for some action $a \in A_k$ and 0 for all other actions, otherwise it is called a *mixed strategy*. A *strategy profile* $\sigma = (\sigma_1, \dots, \sigma_n)$ is a vector of strategies, containing one strategy for each player. If all strategies in σ are pure, the strategy profile corresponds to a joint action $\mathbf{a} \in \mathbb{A}$. An important assumption which is made in normal form games is

that the expected payoffs are linear in the player strategies, i.e. the expected reward for player k for a strategy profile σ is given by:

$$R_k(\sigma) = \sum_{\mathbf{a} \in \mathbb{A}} \prod_{j=1}^n \sigma_j(a_j) R_k(\mathbf{a})$$

with a_j the action for player j in the action profile \mathbf{a} .

14.2.1.2 Types of Games

Depending on the reward functions of the players, a classifications of games can be made. When all players share the same reward function, the game is called a *identical payoff* or *common interest* game. If the total of all players rewards adds up to 0 the game is called a *zero-sum game*. In the latter games wins for certain players translate to losses for other players with opposing goals. Therefore these games are also referred to as pure competitive games. When considering games without special restrictions we speak of *general sum games*.

Table 14.2 Examples of 2-player, 2-action games. From left to right: (a) Matching pennies, a purely competitive (zero-sum) game. (b) The prisoner’s dilemma, a general sum game. (c) The coordination game, a common interest (identical payoff) game. (d) Battle of the sexes, a coordination game where agents have different preferences) Pure Nash equilibria are indicated in bold.

$\begin{array}{c cc} & a1 & a2 \\ \hline a1 & (1,-1) & (-1,1) \\ a2 & (-1,1) & (1,-1) \end{array}$	$\begin{array}{c cc} & a1 & a2 \\ \hline a1 & (5,5) & (0,10) \\ a2 & (10,0) & (\mathbf{1},\mathbf{1}) \end{array}$	$\begin{array}{c cc} & a1 & a2 \\ \hline a1 & (\mathbf{5},\mathbf{5}) & (0,0) \\ a2 & (0,0) & (\mathbf{10},\mathbf{10}) \end{array}$	$\begin{array}{c cc} & a1 & a2 \\ \hline a1 & (\mathbf{2},\mathbf{1}) & (0,0) \\ a2 & (0,0) & (\mathbf{1},\mathbf{2}) \end{array}$
(a)	(b)	(c)	(d)

Examples of these game types can be seen in Table 14.2. The first game in this table, named matching pennies, is an example of a strictly competitive game. This game describes a situation where the two players must each, individually, select one side of a coin to show (i.e. Heads or Tails). When both players show the same side, player one wins and is paid 1 unit by player 2. When the coins do not match, player 2 wins and receives 1 unit from player 1. Since both players are betting against each other, one player's win automatically translates in the other player's loss, therefore this is a zero-sum game.

The second game in Table 14.2, called the prisoner's dilemma, is a general sum game. In this game, 2 criminals have been apprehended by the police for committing a crime. They both have 2 possible actions: cooperate with each other and deny the crime (action a1), or defect and betray the other, implicating him in the crime (action a2). If both cooperate and deny the crime, the police have insufficient evidence and they get a minimal sentence, which translates to a payoff of 5 for both. If one player cooperates, but the other one defects, the cooperator takes all the blame

(payoff 0), while the defector escapes punishment (payoff 10). Should both players defect, however, they both receive a large sentence (payoff 1). The issue in this game is that the cooperate action is *strictly dominated* by the defect action: no matter what action the other player chooses, to defect always gives the highest payoff. This automatically leads to the $(\text{defect}, \text{defect})$ outcome, despite the fact that both players could simultaneously do better by both playing cooperate.

The third game in Table 14.2 is a common interest game. In this case both players receive the same payoff for each joint action. The challenge in this game is for the players to coordinate on the optimal joint action. Selecting the wrong joint action gives a suboptimal payoff and failing to coordinate results in a 0 payoff.

The fourth game, Battle of the sexes, is another example of a coordination game. Here however, the players get individual rewards and prefer different outcomes. Agent 1 prefers (a_1, a_1) , whereas agent 2 prefers (a_2, a_2) . In addition to the coordination problem, the players now also have to agree on which of the preferred outcomes.

Of course games are not restricted to only two actions but can have any number of actions. In Table 14.3 we show some 3-action common interest games. In the first, the climbing game from (Claus and Boutilier, 1998), the Nash equilibria are surrounded by heavy penalties. In the second game, the penalties are left as a parameter $k < 0$. The smaller k , the more difficult it becomes to agree through learning on the preferred solution $((a_1, a_1) \text{ and } (a_3, a_3))$ (The dynamics of these games using a value-iteration approach are analyzed in (Claus and Boutilier, 1998), see also Section 14.2.2).

Table 14.3 Examples of 2-player, 3-action games. From left to right: (a) Climbing game (b) The penalty game, where $k \leq 0$. Both games are of the common interest type. Pure Nash equilibria are indicated in bold.

	a1	a2	a3		a1	a2	a3
a1	(11,11)	(-30,-30)	(0,0)	a1	(10,10)	(0,0)	(k,k)
a2	(-30,-30)	(7,7)	(6,6)	a2	(0,0)	(2,2)	(0,0)
a3	(0,0)	(0,0)	(5,5)	a3	(k,k)	(0,0)	(10,10)
(a)						(b)	

14.2.1.3 Solution Concepts in Games

Since players in a game have individual reward functions which are dependent on the actions of other players, defining the desired outcome of a game is often not clearcut. One cannot simply expect participants to maximize their payoffs, as it may not be possible for all players to achieve this goal at the same time. See for example the Battle of the sexes game in Table 14.2(d).

An important concept for such learning situations, is that of a *best response*. When playing a best response, a player maximizes his payoff with respect to the current strategies of his opponents in the game. That is, it is not possible for the player to improve his reward if the other participants in the game keep their strategies fixed. Formally, we can define this concept as follows:

Definition 14.2. Let $\sigma = (\sigma_1, \dots, \sigma_n)$ be a strategy profile and let σ_{-k} denote the same strategy profile but without the strategy σ_k of player k . A strategy $\sigma_k^* \in \mu(A_k)$ is then called a best response for player k , if following holds:

$$R_k(\sigma_{-k} \cup \sigma_k^*) \geq R_k(\sigma_{-k} \cup \sigma'_k) \quad \forall \sigma'_k \in \mu(A_k)$$

where $\sigma_{-k} \cup \sigma'_k$ denotes the strategy profile where all agents play the same strategy as they play in σ except agent k who plays σ'_k , i.e. $(\sigma_1, \dots, \sigma_{k-1}, \sigma'_k, \sigma_{k+1}, \dots, \sigma_n)$.

A central solution concept in games, is the *Nash equilibrium* (NE). In a Nash equilibrium, the players all play mutual best replies, meaning that each player uses a best response to the current strategies of the other players. Nash (Nash, 1950) proved that every normal form game has at least 1 Nash equilibrium, possibly in mixed strategies. Based on the concept of best response we can define a Nash equilibrium as:

Definition 14.3. A strategy profile $\sigma = (\sigma_1, \dots, \sigma_n)$ is called a Nash equilibrium if for each player k , the strategy σ_k is a best response to the strategies of the other players σ_{-k} .

Thus, when playing a Nash equilibrium, no player in the game can improve his payoff by unilaterally deviating from the equilibrium strategy profile. As such no player has an incentive to change his strategy, and multiple players have to change their strategy simultaneously in order to escape the Nash equilibrium.

In common interest games such as the coordination in Table 14.2(c), the Nash equilibrium corresponds to a local optimum for all players, but it does not necessarily correspond to the global optimum. This can clearly be seen in the coordination game, where we have 2 Nash equilibria: the play $(a1, a1)$ which gives both players a reward of 5 and the global optimum $(a2, a2)$ which results in a payoff of 10.

The prisoner's dilemma game in Table 14.2 shows that a Nash equilibrium does not necessarily correspond to the most desirable outcome for all agents. In the unique Nash equilibrium both players prefer the 'defect' action, despite the fact that both would receive when both are cooperating. The cooperative outcome is not a Nash equilibrium, however, as in this case both players can improve their payoff by switching to the 'defect' action.

The first game, matching pennies, does not have a pure strategy Nash equilibrium, as no pure strategy is a best response to another pure best response. Instead the Nash equilibrium for this game is for both players to choose both sides with equal probability. That is, the Nash strategy profile is $((1/2, 1/2), (1/2, 1/2))$.

14.2.2 Reinforcement Learning in Repeated Games

The games described above are often used as test cases for multi-agent reinforcement learning techniques. Unlike in the game theoretical setting, agents are not assumed to have full access to the payoff matrix. In the reinforcement learning setting, agents are taken to be players in a normal form game, which is played repeatedly, in order to improve their strategy over time.

It should be noted that these *repeated games* do not yet capture the full multi-agent reinforcement learning problem. In a repeated game all changes in the expected reward are due to changes in strategy by the players. There is no changing environment state or state transition function external to the agents. Therefore, repeated games are sometimes also referred to as *stateless games*. Despite this limitation, we will see further in this section that these games can already provide a challenging problem for independent learning agents, and are well suited to test coordination approaches. In the next section, we will address the Markov game framework which does include a dynamic environment.

A number of different considerations have to be made when dealing with reinforcement learning in games. As is common in RL research, but contrary to traditional economic game theory literature, we assume that the game being played is initially unknown to the agents, i.e. agents do not have access to the reward function and do not know the expected reward that will result from playing a certain (joint) action. However, RL techniques can still differ with respect to the observations the agents make. Moreover, we also assume that the game payoffs can be stochastic, meaning that a joint action does not always result in the same deterministic reward for each agent. Therefore, actions have to be sampled repeatedly.

Since expected rewards depend on the strategy of all agents, many multi-agent RL approaches assume that the learner can observe the actions and/or rewards of all participants in the game. This allows the agent to model its opponents and to explicitly learn estimates over joint actions. It could be argued however, that this assumption is unrealistic, as in multi-agent systems which are physically distributed this information may not be readily available. In this case the RL techniques must be able to deal with the non-stationary rewards caused by the influence of the other agents. As such, when developing a multi-agent reinforcement learning application it is important to consider the information available in a particular setting in order to match this setting with an appropriate technique.

14.2.2.1 Goals of Learning

Since it is in general impossible for all players in a game to maximize their payoff simultaneously, most RL methods attempt to achieve Nash equilibrium play. However, a number of criticisms can be made of the Nash equilibrium as a solution concept for learning methods. The first issue is that Nash equilibria need not be unique, which leads to an equilibrium selection problem. In general, multiple Nash equilibria can exist for a single game. These equilibria can also differ in the

payoff they give to the players. This means that a method learning Nash equilibria, cannot guarantee a unique outcome or even a unique payoff for the players. This can be seen in the coordination game of Table 14.2(c), where 2 Nash equilibria exist, one giving payoff 5 to the agents, and the other giving payoff 10. The game in Table 14.3(b) also has multiple NE, with (a_1, a_1) and (a_3, a_3) being the 2 optimal ones. This results in a coordination problem for learning agents, as both these NE have the same quality.

Furthermore, since the players can have different expected payoffs even in an equilibrium play, the different players may also prefer different equilibrium outcomes, which means that care should be taken to make sure the players coordinate on a single equilibrium. This situation can be observed in the Battle of the sexes game in Table 14.2(d), where 2 pure Nash equilibria exist, but each player prefers a different equilibrium outcome.

Another criticism is that a Nash Equilibrium does not guarantee optimality. While playing a Nash equilibrium assures that no single player can improve his payoff by unilaterally changing its strategy, it does not guarantee that the players globally maximize their payoffs, or even that no play exists in which the players simultaneously do better. It is possible for a game to have non-Nash outcomes, which nonetheless result in a higher payoff to all agents than they would receive for playing a Nash equilibrium. This can be seen for example in the prisoner's dilemma in Table 14.2(c).

While often used as the main goal of learning, Nash equilibria are not the only possible solution concept in game theory. In part due to the criticisms mentioned above, a number of alternative solution concepts for games have been developed. These alternatives include a range of other equilibrium concepts, such as the *Correlated Equilibrium(CE)*(Aumann, 1974), which generalizes the Nash equilibrium concept, or the *Evolutionary Stable Strategy (ESS)*(Smith, 1982), which refines the Nash equilibrium. Each of these equilibrium outcomes has its own applications and (dis)advantages. Which solution concept to use depends on the problem at hand, and the objective of the learning algorithm. A complete discussion of possible equilibrium concepts is beyond the scope of this chapter. We focus on the Nash equilibrium and briefly mention regret minimization as these are the approaches most frequently observed in the multi-agent learning literature. A more complete discussion of solution concepts can be found in many textbooks, e.g. (Leyton-Brown and Shoham, 2008).

Before continuing, we mention one more evaluation criterion, which is regularly used in repeated games: the notion of *regret*. Regret is the difference between the payoff an agent realized and the maximum payoff the agent could have obtained using some fixed strategy. Often the fixed strategies that one compares the agent performance to, are simply the pure strategies of the agent. In this case, the total regret of the agent is the accumulated difference between the obtained reward and the reward the agent would have received for playing some fixed action. For an agent k , given the history of play at time T , this is defined as:

$$\mathcal{R}_T = \max_{a \in A_k} \sum_{t=1}^T R_k(\mathbf{a}_{-k}(t) \cup \{a\}) - R_k(\mathbf{a}(t)), \quad (14.1)$$

where $\mathbf{a}(t)$ denotes the joint action played at time t and $\mathbf{a}_{-k}(t) \cup \{a\}$ denotes the same joint action but with player k playing action a . Most regret based learning approaches attempt to minimize the average regret \mathcal{R}_T/T of the learner. Exact calculation of this regret requires knowledge of the reward function and observation of the actions of other agents in order to determine the $R_k(\mathbf{a}_{-k}(t) \cup \{a\})$ term. If this information is not available, regret has to be estimated from previous observations. Under some assumptions regret based learning can be shown to converge to some form of equilibrium play (Foster and Young, 2003; Hart and Mas-Colell, 2001).

14.2.2.2 Q-Learning in Games

A natural question to ask is what happens when agents use a standard, single-agent RL technique to interact in a game environment. Early research into multi-agent RL focussed largely on the application of Q-learning to repeated games. In this so called independent or uninformed setting, each player k keeps an individual vector of estimated Q-values $Q_k(a)$, $a \in A_k$. The players learn Q-values over their own action set and do not use any information on other players in the game. Since there is no concept of environment state in repeated games, a single vector of estimates is sufficient, rather than a full table of state-action pairs, and the standard Q-learning update is typically simplified to its stateless version:

$$Q(a) \leftarrow Q(a) + \alpha[r(t) - Q(a)] \quad (14.2)$$

In (Claus and Boutilier, 1998) the dynamics of stateless Q-learning in repeated normal form common interest games are empirically studied. The key questions here are: is simple Q-learning still guaranteed to converge in a multi-agent setting, and if so, does it converge to (the optimal) equilibrium. It also relates independent Q-learners to joint action learners (see below) and investigates how the rates of convergence and limit points are influenced by the game structures and action selection strategies. In a related branch of research (Tuyls and Nowé, 2005; Wunder et al, 2010) the dynamics of independent Q-learning are studied using techniques from *evolutionary game theory* (Smith, 1982).

While independent Q-learners were shown to reach equilibrium play under some circumstances, they also demonstrated a failure to coordinate in some games, and even failed to converge altogether in others.

They compared joint action learners to independent learners. In the former the agents learn Q-values for all joint actions, with other words each agent j learns a Q-value for all a in A . The action selection is done by each agent individually based on the belief the agents has about the other agents strategy. Equation 14.3 expresses that the Q-value of the joint action is weighted according to the probability the other agents will select some particular value. The Expected Values (EV) can then be used in combination with any action selection technique. Claus and Boutilier showed

experimentally using the games of table 2, that joint action learners and independent learners using a Boltzmann exploration strategy with decreasing temperature behave very similar. These learners have been studied from an evolutionary game theory point of view in (Tuyls and Nowé, 2005) and it has been shown that these learners will converge to evolutionary stable NE which are not necessarily pareto optimal.

$$EV(a^i) = \sum_{a^{-i} \in A_{-i}} Q(a^{-i} \cup \{a^i\}) \prod_{j \neq i} \{Pr_{a^{-i}[j]}^i\} \quad (14.3)$$

However the learners have difficulties to reach the optimal NE, and more sophisticated exploration strategies are needed to increase the probability of converging to the optimal NE. The reason that simple exploration strategies are not sufficient is mainly due to the fact that the actions involved in the optimal NE often lead to much lower payoff when combined with other actions, the potential quality of the action is therefore underestimated. For example in game 2a the action a1 of the row player, will only lead to the highest reward 11 when combined with action a1 of the column player. During the learning phase, agents are still exploring and action a1 will also be combined with actions a2 and a3. As a result the agents will often settle for the more “safe” NE (a2,a2). A similar behavior is observed in game 2b, since miscoordination on the 2 NE is punished, the bigger the penalty ($k;0$) the more difficult it becomes for the agents to reach either of the optimal NE. This also explains why independent learners are generally not able to converge to a NE when they are allowed to use any, including a random exploration strategy. Whereas in single agent Q-learning, the particular exploration strategy does not affect the eventual convergence (Tsitsiklis, 1994) this no longer holds in a MAS setting.

The limitations of single-agent Q-learning have led to a number of extensions of the Q-learning algorithm for use in repeated games. Most of these approaches focus on coordination mechanisms allowing Q-learners to reach the optimal outcome in common interest games. The frequency maximum Q-learning (FMQ) algorithm (Kapetanakis and Kudenko, 2002), for example, keeps a frequency value $freq(R^*, a)$ indicating how often the maximum reward so far (R^*) has been observed for a certain action a . This value is then used as a sort of heuristic which is added to the Q-values. Instead of using Q-values directly, the FMQ algorithm relies on following heuristic evaluation of the actions:

$$EV(a) = Q(a) + w.freq(R^*, a).R^*, \quad (14.4)$$

where w is a weight that controls the importance of the heuristic value $freq(R^*, a).R^*$. The algorithm was empirically shown to be able to drive learners to the optimal joint action in common interest games with deterministic payoffs.

In (Kapetanakis et al, 2003) the idea of commitment sequences has been introduced to allow independent learning in games with stochastic payoffs. A commitment sequence is a list of time slots for which an agent is committed to selecting always the same action. These sequences of time slots are generated according to some protocol the agents are aware of. Using this guarantee that at time slots belonging

to the same sequence the agents are committed to always select the same individual action, the agents are able to distinguish between the two sources of uncertainty: the noise on the reward signal and the influence on the reward by the actions taken by the other agents. This allows the agents to deal with games with stochastic payoffs.

A recent overview of multi-agent Q-learning approaches can be found in (Wunder et al, 2010).

14.2.2.3 Gradient Ascent Approaches

As an alternative to the well known Q-learning algorithm, we now list some approaches based on gradient following updates. We will focus on players that employ learning automata (LA) reinforcement schemes. Learning automata are relatively simple policy iterators, that keep a vector action probabilities \mathbf{p} over the action set A . As is common in RL, these probabilities are updated based on a feedback received from the environment. While initial studies focussed mainly on a single automaton in n-armed bandit settings, RL algorithms using multiple automata were developed to learn policies in MDPs (Wheeler Jr and Narendra, 1986). The most commonly used LA update scheme is called Linear Reward-Penalty and updates the action probabilities as follows:

$$p_i(t+1) = p_i(t) + \lambda_1 b(t)(1 - p_i(t)) - \lambda_2(1 - r(t))p_i(t) \quad (14.5)$$

if $a(t) = a_i$,

$$p_j(t+1) = p_j(t) - \lambda_1 r(t)p_j(t) + \lambda_2(1 - r(t))\left(\frac{1}{K-1} - p_j(t)\right) \quad (14.6)$$

if $a_j \neq a_i$,

$r(t)$ being the feedback received at time t and K the number of actions in available to the auomaton. λ_1 and λ_2 are constants, called the reward and penalty parameter respectively. Depending on the values of these parameters 3 distinct variations of the algorithm can be considered. When $\lambda_1 = \lambda_2$, the algorithm is referred to as *Linear Reward-Penalty* (L_{R-P}) while it is called *Linear Reward- ϵ Penalty* ($L_{R-\epsilon P}$) when $\lambda_1 >> \lambda_2$. If $\lambda_2 = 0$ the algorithm is called *Linear Reward-Inaction* (L_{R-I}). In this case, λ_1 is also sometimes called the learning rate:

$$p_i(t+1) = p_i(t) + \lambda_1 r(t)(1 - p_i(t)) \quad (14.7)$$

if $a(t) = a_i$

$$p_j(t+1) = p_j(t) - \lambda_1 r(t)p_j(t) \quad (14.8)$$

if $a_j \neq a_i$

This algorithm has also been shown to be a special case of the REINFORCE (Williams, 1992) update rules. Despite the fact that all these update rules are derived

from the same general scheme, they exhibit very different learning behaviors. Interestingly, these learning schemes perform well in game contexts, even though they do not require any information (actions, rewards, strategies) on the other players in the game. Each agent independently applies a LA update rule to change the probabilities over its actions. Below we list some interesting properties of LA in game settings. In two-person zero-sum games, the L_{R-I} scheme converges to the Nash equilibrium when this exists in pure strategies, while the $L_{R-\varepsilon P}$ scheme is able to approximate mixed equilibria. In n -player common interest games reward-inaction also converges to a pure Nash equilibrium. In (Sastry et al, 1994), the dynamics of reward-inaction in general sum games are studied. The authors proceed by approximating the update in the automata game by a system of ordinary differential equations. Following properties are found to hold for the L_{R-I} dynamics:

- All Nash equilibria are stationary points.
- All strict Nash equilibria are asymptotically stable.
- All stationary points that are not Nash equilibria are unstable.

Furthermore, in (Verbeeck, 2004) it is shown that an automata team using the reward-inaction scheme will converge to a pure joint strategy with probability 1 in common as well as conflicting interest games with stochastic payoffs. These results together imply local convergence towards pure Nash equilibria in n -player general-sum games (Verbeeck, 2004). Since NE with higher payoffs are stronger attractors for the LA, the agents are more likely to reach the better NE. Equipped with an exploration strategy with only requires very limited communications, the agents are able to explore the interesting NE without the need for exhaustive exploration and once these are found, different solution concepts can be considered, for example fair solutions alternating between different Pareto optimal solutions.

In (Verbeeck et al, 2005) it has also been shown that these LA based approach is able to converge in a setting where agents take actions asynchronously and the rewards are delayed as is common in load balancing settings or congestion games.

Another gradient technique frequently studied in games is the *Infinitesimal Gradient Ascent* (IGA) family of algorithms (Singh et al, 2000; Bowling and Veloso, 2001; Zinkevich, 2003; Bowling, 2005). In addition to demonstrating Nash equilibrium convergence in a number of repeated game settings, several of these papers also evaluate the algorithms with respect to their regret.

14.3 Sequential Games

While highlighting some of the important issues introduced by learning in a multi-agent environment, the traditional game theory framework does not capture the full complexity of multi-agent reinforcement learning. An important part of the reinforcement learning problem is that of making sequential decisions in an environment with state transitions and cannot be described by standard normal form games, as they allow only stationary, possibly stochastic, reward functions that depend solely

on the actions of the players. In a normal form game there is no concept of a system with state transitions, a central issue of the Markov decision process concept. Therefore, we now consider a richer framework which generalizes both repeated games and MDPs. Introducing multiple agents to the MDP model significantly complicates the problem that the learning agents face. Both rewards and transitions in the environment now depend on the actions of all agents present in the system. Agents are therefore required to learn in a joint action space. Moreover, since agents can have different goals, an optimal solution which maximizes rewards for all agents simultaneously may fail to exist.

To accommodate the increased complexity of this problem we use the representation of Stochastic of Markov games (Shapley, 1953). While they were originally introduced in game theory as an extension of normal form games, Markov games also generalize the Markov Decision process and were more recently proposed as the standard framework for multi-agent reinforcement learning (Littman, 1994). As the name implies, Markov games still assume that state transitions are Markovian, however, both transition probabilities and expected rewards now depend on the joint action of all agents. Markov games can be seen as an extension of MDPs to the multi-agent case, and of repeated games to multiple state case. If we assume only 1 agent, or the case where other agents play a fixed policy, the Markov game reduces to an MDP. When the Markov game has only 1 state, it reduces to a repeated normal form game.

14.3.1 Markov Games

An extension of the single agent Markov decision process (MDP) to the multi-agent case can be defined by Markov Games. In a Markov Game, joint actions are the result of multiple agents choosing an action independently.

Definition 14.4. A Markov game is a tuple $(n, S, A_1, \dots, n, R_1, \dots, n, T)$:

- n the number of agents in the system.
- $S = \{s^1, \dots, s^N\}$ a finite set of system states.
- A_k the action set of agent k .
- $R_k : S \times A_1 \times \dots \times A_n \times S \rightarrow \mathbb{R}$, the reward function of agent k .¹
- $T : S \times A_1 \times \dots \times A_n \rightarrow \mu(S)$ the transition function.

Note that $A_k(s^i)$ is now the action set available to agent k in state s^i , with $k : 1 \dots n$, n being the number of agents in the system and $i : 1, \dots, N$, N being the number of states in the system. Transition probabilities $T(s^i, \mathbf{a}^i, s^j)$ and rewards $R^k(s^i, \mathbf{a}^i, s^j)$ now depend on a current state s^i , next state s^j and a joint action from state s^i , i.e. $\mathbf{a}^i = (a_1^i, \dots, a_n^i)$ with $a_k^i \in A_k(s^i)$. The reward function $R_k(s^i, \mathbf{a}^i, s^j)$ is now individual

¹ As was the case for MDPs, one can also consider the equivalent case where reward does not depend on the next state.

to each agent k . Different agents can receive different rewards for the same state transition. Transitions in the game are again assumed to obey the Markov property.

As was the case in MDPs, agents try to optimize some measure of their future expected rewards. Typically they try to maximize either their future discounted reward or their average reward over time. The main difference with respect to single agent RL, is that now these criteria also depend on the policies of other agents. This results in the following definition for the expected discounted reward for agent k under a joint policy $\pi = (\pi_1, \dots, \pi_n)$, which assigns a policy π_i to each agent i :

$$V_k^\pi(s) = E^\pi \left\{ \sum_{t=0}^{\infty} \gamma^t r_k(t+1) \mid s(0) = s \right\} \quad (14.9)$$

while the average reward for agent k under this joint policy is defined as:

$$J_k^\pi(s) = \lim_{T \rightarrow \infty} \frac{1}{T} E^\pi \left\{ \sum_{t=0}^{T-1} r_k(t+1) \mid s(0) = s \right\} \quad (14.10)$$

Since it is in general impossible to maximize this criterion for all agents simultaneously, as agents can have conflicting goals, agents playing a Markov game face the same coordination problems as in repeated games. Therefore, typically one relies again on equilibria as the solution concept for these problems. The best response and Nash equilibrium concepts can be extended to Markov games, by defining a policy π_k as a best response, when no other policy for agent k exists which gives a higher expected future reward, provided that the other agents keep their policies fixed.

It should be noted that learning in a Markov game introduces several new issues over learning in MDPs with regard to the policy being learned. In an MDP, it is possible to prove that, given some basic assumptions, an optimal deterministic policy always exists. This means it is sufficient to consider only those policies which deterministically map each state to an action. In Markov games, however, where we must consider equilibria between agent policies, this no longer holds. Similarly to the situation in repeated games, it is possible that a discounted Markov game, only has Nash equilibria in which stochastic policies are involved. As such, it is not sufficient to let agents map a fixed action to each state: they must be able to learn a mixed strategy. The situation becomes even harder when considering other reward criteria, such as the average reward, since then it is possible that no equilibria in stationary strategies exist (Gillette, 1957). This means that in order to achieve an equilibrium outcome, the agents must be able to express policies which condition the action selection in a state on the entire history of the learning process. Fortunately, one can introduce some additional assumptions on the structure of the problem to ensure the existence of stationary equilibria (Sobel, 1971).

14.3.2 Reinforcement Learning in Markov Games

While in normal form games the challenges for reinforcement learners originate mainly from the interactions between the agents, in Markov games they face the

additional challenge of an environment with state transitions. This means that the agents typically need to combine coordination methods or equilibrium solvers used in repeated games with MDP approaches from single-agent RL.

14.3.2.1 Value Iteration

A number of approaches have been developed, aiming at extending the successful Q-learning algorithm to multi-agent systems. In order to be successful in a multi-agent context, these algorithms must first deal with a number of key issues.

Firstly, immediate rewards as well as the transition probabilities depend on the actions of all agents. Therefore, in a multi-agent Q-learning approach, the agent does not simply learns to estimate $Q(s,a)$ for each state action pair, but rather estimates $Q(s,\mathbf{a})$ giving the expected future reward for taking the joint action $\mathbf{a} = a_1, \dots, a_n$ in state s . As such, contrary to the single agent case, the agent does not have a single estimate for the future reward it will receive for taking an action a_k in state s . Instead, it keeps a vector of estimates, which give the future reward of action a_k , depending on the joint action \mathbf{a}_{-k} played by the other agents. During learning, the agent selects an action and then needs to observe the actions taken by other agents, in order to update the appropriate $Q(s,\mathbf{a})$ value.

This brings us to the second issue that a multi-agent Q-learner needs to deal with: the state values used in the bootstrapping update. In the update rule of single agent Q-learning the agent uses a maximum over its actions in the next state s' . This gives the current estimate of the value of state s' under the greedy policy. But as was mentioned above, the agent cannot predict the value of taking an action in the next state, since this value also depends on the actions of the other agents. To deal with this problem, a number of different approaches have been developed which calculate a value of state s' by also taking into account the other agents. All these algorithms, of which we describe a few examples below, correspond to the general multi-agent Q-learning template given in Algorithm 23, though each algorithm differs in the method used to calculate the $V_k(s')$ term in the Q-learning update.

A first possibility to determine the expected value of a state $V_k(s)$ is to employ *opponent modeling*. If the learning agent is able to estimate the policies used by the other agents, it can use this information to determine the expected probabilities with which the different joint actions are played. Based on these probabilities the agent can then determine the expected value of a state. This is the approach followed, for example, by the Joint Action Learner (JAL) algorithm (Claus and Boutilier, 1998). A joint action learner keeps counts $c(s, \mathbf{a}_{-k})$ of the number of times each state joint action pair (s, \mathbf{a}_{-k}) with $\mathbf{a}_{-k} \in \mathbb{A}_{-k}$ is played. This information can then be used to determine the empirical frequency of play for the possible joint actions of the other agents:

$$F(s, \mathbf{a}_{-k}) = \frac{c(s, \mathbf{a}_{-k})}{\sum_{\mathbf{a}_{-k}' \in \mathbb{A}_{-k}} n(s, \mathbf{a}_{-k}')},$$

```

t=0
Qk(s,a) = 0 ∀s,a,k
repeat
  for all agents k do
    select action ak(t)
    execute joint action a = (a1,...,an)
    observe new state s', rewards rk
    for all agents k do
      Qk(s,a) = Qk(s,a) + α [Rk(s,a) + γVk(s') - Qk(s,a)]
  until Termination Condition

```

Algorithm 23. Multi-Agent Q-Learning

This estimated frequency of play for the other agents, allows the joint action learner to calculate the expected Q-values for a state:

$$V_k(s) = \max_{a_k} Q(s, a_k) = \sum_{\mathbf{a}_{-k} \in \mathbb{A}_{-k}} F(s, \mathbf{a}_{-k}) \cdot Q(s, a_k, \mathbf{a}_{-k}),$$

where $Q(s, a_k, \mathbf{a}_{-k})$ denotes the Q-value in state s for the joint action in which agent k plays a_k and the other agents play according to \mathbf{a}_{-k} . These expected Q-values can then be used for the agent's action selection, as well as in the Q-learning update, just as in the standard single-agent Q-learning algorithm.

Another method used in multi-agent Q-learning is to assume that the other agents will play according to some strategy. For example, in the minimax Q-learning algorithm (Littman, 1994), which was developed for 2-agent zero-sum problems, the learning agent assumes that its opponent will play the action which minimizes the learner's payoff. This means that the max operator of single agent Q-learning is replaced by the minimax value:

$$V_k(s) = \min_{a'} \max_{\sigma \in \mu(A)} \sum_{a \in A} \sigma(a) Q(s, a, a')$$

The Q-learning agent maximizes over its strategies for state s , while assuming that the opponent will pick the action which minimizes the learner's future rewards. Note that the agent does not just maximizes over the deterministic strategies, as it is possible that the maximum will require a mixed strategy. This system was later generalized to *friend-or-foe* Q-learning (Littman, 2001a), where the learning agent deals with multiple agents by marking them either as friends, who assist to maximize its payoff or foes, who try to minimize the payoff.

Alternative approaches assume that the agents will play an equilibrium strategy. For example, Nash-Q (Hu and Wellman, 2003) observes the rewards for all agents and keeps estimates of Q-values not only for the learning agent, but also for all other agents. This allows the learner to represent the joint action selection in each state as a game, where the entries in the payoff matrix are defined by the Q-values of the agents for the joint action. This representation is also called the *stage game*.

A Nash-Q agent then assumes that all agents will play according to a Nash equilibrium of this stage game in each state:

$$V_k(s) = \text{Nash}_k(s, Q_1, \dots, Q_n),$$

where $\text{Nash}_k(s, Q_1, \dots, Q_n)$ denotes the expected payoff for agent k when the agents play a Nash equilibrium in the stage game for state s with Q-values Q_1, \dots, Q_n . Under some rather strict assumptions on the structure of the stage games, Nash-Q can be shown to converge in self-play to a Nash equilibrium between agent policies.

The approach used in Nash-Q can also be combined with other equilibrium concepts, for example correlated equilibria (Greenwald et al, 2003) or the Stackelberg equilibrium (Kononen, 2003). The main difficulty with these approaches is that the value is not uniquely defined when multiple equilibria exist, and coordination is needed to agree on the same equilibrium. In these cases, additional mechanisms are typically required to select some equilibrium.

While the intensive research into value iteration based multi-agent RL has yielded some theoretical guarantees (Littman, 2001b), convergence results in the general Markov game case remain elusive. Moreover, recent research indicates that a reliance on Q-values alone may not be sufficient to learn an equilibrium policy in arbitrary general sum games (Zinkevich et al, 2006) and new approaches are needed.

14.3.2.2 Policy Iteration

In this section we describe policy iteration for multi-agent reinforcement learning. We focus on an algorithm called Interconnected Learning Automata for Markov Games (MG-ILA)(Vrancx et al, 2008b), based on the learning automata from Section 14.2.2.3 and which can be applied to average reward Markov games. The algorithm can be seen as an implementation of the *actor-critic* framework, where the policy is stored using learning automata. The main idea is straightforward: each agent k puts a single learning automaton LA (k, i) in each system state s^i . At each time step only the automata of the current state are active. Each automaton then individually selects an action for its corresponding agent. The resulting joint action triggers the next state transition and immediate rewards. Automata are not updated using immediate rewards but rather using a response estimating the average reward. The complete algorithm is listed in Algorithm 24.

An interesting aspect of this algorithm is that its limiting behavior can be approximated by considering a normal form game in which all the automata are players. A play in this game selects an action for each agent in each state, and as such corresponds to a pure, joint policy for all agents. Rewards in the game are the expected average rewards for the corresponding joint policies. In (Vrancx et al, 2008b), it is shown that the algorithm will converge to a pure Nash equilibrium in this resulting game (if it exists), and that this equilibrium corresponds to a pure equilibrium between the agent policies. The game approximation also enables an evolutionary game theoretic analysis of the learning dynamics (Vrancx et al, 2008a), similar to that applied to repeated games.

```

initialise  $r_{prev}(s,k)$ ,  $t_{prev}(s)$ ,  $a_{prev}(s,k)$ ,  $r_{tot}(k)$ ,  $\rho_k(s,a)$ ,  $\eta_k(s,a)$  to zero,  $\forall s, k, a$ .
 $s \leftarrow s(0)$ 
loop
  for all Agents  $k$  do
    if  $s$  was visited before then
      • Calculate received reward and time passed since last visit to state  $s$ :
        
$$\Delta r_k = r_{tot}(k) - r_{prev}(s,k)$$

        
$$\Delta t = t - t_{prev}(s)$$

      • Update estimates for action  $a_{prev}(s,k)$  taken on last visit to  $s$ :
        
$$\rho_k(s, a_{prev}(s,k)) = \rho_k(s, a_{prev}(s,k)) + \Delta r_k$$

        
$$\eta_k(s, a_{prev}(s,k)) = \eta_k(s, a_{prev}(s,k)) + \Delta t$$

      • Calculate feedback:
        
$$\beta_k(t) = \frac{\rho_k(s, a_{prev}(s,k))}{\eta_k(s, a_{prev}(s,k))}$$

      • Update automaton  $LA(s,k)$  using  $L_{R-I}$  update with  $a(t) = a_{prev}(s,k)$  and  $\beta_k(t)$  as above.
      • Let  $LA(s,k)$  select an action  $a_k$ .
    • Store data for current state visit:
      
$$t_{prev}(s) \leftarrow t$$

      
$$r_{prev}(s,k) \leftarrow r_{tot}(k)$$

      
$$a_{prev}(s,k) \leftarrow a_k$$

    • Execute joint action  $\mathbf{a} = (a_1, \dots, a_n)$ , observe immediate rewards  $r_k$  and new state  $s'$ 
    •  $s \leftarrow s'$ 
    •  $r_{tot}(k) \leftarrow r_{tot}(k) + r_k$ 
    •  $t \leftarrow t + 1$ 

```

Algorithm 24. MG-ILA

While not as prevalent as value iteration based methods, a number of interesting approaches based on policy iteration have been proposed. Like the algorithm described above, these methods typically rely on a gradient based search of the policy space. (Bowling and Veloso, 2002), for example, proposes an actor-critic framework which combines tile coding generalization with policy gradient ascent and uses the Win or Learn Fast (WoLF) principle. The resulting algorithm is empirically shown to learn in otherwise intractably large problems. (Kononen, 2004) introduces a policy gradient method for common-interest Markov games which extends the single agent methods proposed by (Sutton et al, 2000). Finally, (Peshkin et al, 2000)

develop a gradient based policy search method for partially observable, identical payoff stochastic games. The method is shown to converge to local optima which are, however, not necessarily Nash equilibria between agent policies.

14.4 Sparse Interactions in Multi-agent System

A big drawback of reinforcement learning in Markov games is the size of the state-action-space in which the agents are learning. All agents learn in the entire joint state-action space and as such these approaches become quickly infeasible for all but the smallest environments and with a limited number of agents. Recently, a lot of attention has gone into mitigating this problem. The main intuition for these approaches is to only explicitly consider the other agents if a better payoff can be obtained by doing so. In all other situations the other agents can safely be ignored and as such have the advantages of learning in a small state-action space, while also having access to the necessary information to deal with the presence of other agents, if this is beneficial. An example of such systems is an automated warehouse, where the automated guided vehicles only have to consider each other when they are close by to each other. We can distinguish two different lines of research: agents can base their decision for coordination on the actions that are selected, or agents can focus on the state information at their disposal, and learn when it is beneficial to observe the state information of other agents. We will describe both these approaches separately in Sections 14.4.2.1 and 14.4.2.2

14.4.1 Learning on Multiple Levels

Learning with sparse interactions provides an easy way of dealing with the exponential growth of the state space in terms of the number of agents involved in the learning process. Agents should only rely on more global information, in those situations where the transition of the state of the agent to the next state and the rewards the agents experience are not only dependent on the local state information of the agent performing the action, but also on the state information or actions of other agents. The idea of sparse interactions is '*When is an agent experiencing influence from another agent?*'. Answering this question, allows an agent to know when it can select its actions independently (i.e. the state transition function and reward function are only dependent on its own action) or when it must coordinate with other agents (i.e. the state transition function and the reward function is the effect of the joint action of multiple agents). This leads naturally to a decomposition of the multi-agent learning process into two separate layers. The top layer should learn when it is necessary to observe the state information about other agents and select whether pure independent learning is sufficient, or whether some form of coordination between the agents is required. The bottom layer contains a single agent learning technique, to be used when there is no risk of influence by other agents, and a multi-agent technique, to be used when the state transition and reward the agent receives is dependent of the current state and actions of other agents. Figure 14.2 shows a graphical

representation of this framework. In the following subsection we begin with an overview of algorithms that approach this problem from the action space point of view, and focus on the coordination of actions.

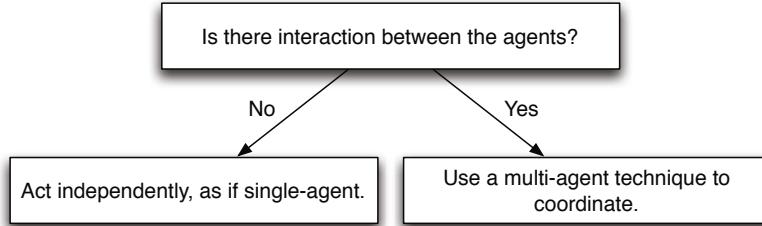


Fig. 14.2 Decoupling the learning process by learning when to take the other agent into account on one level, and acting on the second level

14.4.2 Learning to Coordinate with Sparse Interactions

14.4.2.1 Learning Interdependencies among Agents

Kok & Vlassis proposed an approach based on a sparse representation for the joint action space of the agents while observing the entire joint state space. More specifically they are interested in learning joint-action values for those states where the agents explicitly need to coordinate. In many problems, this need only occurs in very specific situations (Guestrin et al, 2002b). *Sparse Tabular Multiagent Q-learning* maintains a list of states in which coordination is necessary. In these states, agents select a joint action, whereas in all the uncoordinated states they all select an action individually (Kok and Vlassis, 2004b). By replacing this list of states by coordination graphs (CG) it is possible to represent dependencies that are limited to a few agents (Guestrin et al, 2002a; Kok and Vlassis, 2004a, 2006). This technique is known as *Sparse Cooperative Q-learning* (SCQ). Figure 14.3 shows a graphical representation of a simple CG for a situation where the effect of the actions of agent 4 depend on the actions of agent 2 and the actions of agent 2 and 3 both depend on the actions of agent 1, so the nodes represent the agents, while an edge defines a dependency between two agents. If agents transitioned into a coordinated state, they applied a variable elimination algorithm to compute the optimal joint action for the current state. In all other states, the agents select their actions independently.

In later work, the authors introduced *Utile Coordination* (Kok et al, 2005). This is a more advanced algorithm that uses the same idea as SCQ, but instead of having to define the CGs beforehand, they are being learned online. This is done by maintaining statistical information about the obtained rewards conditioned on the states and actions of the other agents. As such, it is possible to learn the context specific dependencies that exist between the agents and represent them in a CG. This technique is however limited to fully cooperative MAS.

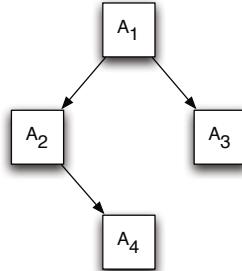


Fig. 14.3 Simple coordination graph. In the situation depicted the effect of the actions of agent 4 depends on the actions of agent 2 and the actions of agent 2 and 3 both depend on the actions of agent 1.

The primary goal of these approaches is to reduce the joint-action space. However, the computation or learning in the algorithms described above, always employ a complete multi-agent view of the entire joint-state space to select their actions, even in states where only using local state information would be sufficient. As such, the state space in which they are learning is still exponential in the number of agents, and its use is limited to situations in which it is possible to observe the entire joint state.

14.4.2.2 Learning a Richer State Space Representation

Instead of explicitly learning the optimal coordinated action, a different approach consists in learning in which states of the environment it is beneficial to include the state information about other agents. We will describe two different methods. The first method learns in which states coordination is beneficial using an RL approach. The second method learns the set of states in which coordination is necessary based on the observed rewards. Unlike the approaches mentioned in Section 14.4.2.1, these approaches can also be applied to conflicting interest games and allow independent action selection.

The general idea of the approaches described in this section are given by Figure 14.4. These algorithms will expand the local state information of an agent to incorporate the information of another agent if this information is necessary to avoid suboptimal rewards.

Learning of Coordination

Spaan and Melo approached the problem of coordination from a different angle than Kok & Vlassis (Spaan and Melo, 2008). They introduced a new model for multi-agent decision making under uncertainty called *interaction-driven Markov games* (IDMG). This model contains a set of interaction states which lists all the states in which coordination is beneficial.

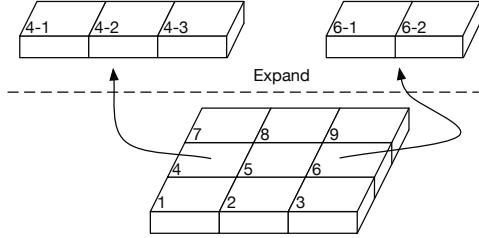


Fig. 14.4 Graphical representation of state expansion with sparse interactions. Independent single states are expanded to joint-states where necessary. Agents begin with 9 independent states. After a while states 4 and 6 of an agent are expanded to include the states of another agent.

In later work, Melo and Veloso introduced an algorithm where agents learn in which states they need to condition their actions on the local state information of other agents (Melo and Veloso, 2009). As such, their approach can be seen as a way of solving an IDMG where the states in which coordination is necessary is not specified beforehand. To achieve this, they augment the action space of each agent with a pseudo-coordination action (COORDINATE). This action will perform an active perception step. This could for instance be a broadcast to the agents to divulge their location or using a camera or sensors to detect the location of the other agents. This active perception step will decide whether coordination is necessary or if it is safe to ignore the other agents. Since the penalty of miscoordination is bigger than the cost of using the active perception, the agents learn to take this action in the interaction states of the underlying IDMG. This approach solves the coordination problem by deferring it to the active perception mechanism.

The active perception step of LoC can consist of the use of a camera, sensory data, or communication to reveal the local state information of another agent. As such the outcome of the algorithm depends on the outcome of this function. Given an adequate active perception function, LoC is capable of learning a sparse set of states in which coordination should occur. Note that depending on the active perception function, this algorithm can be used for both cooperative as conflicting interest systems.

The authors use a variation on the standard Q-learning update rule:

$$Q_k^C(s, a_k) \leftarrow (1 - \alpha(t))Q_k^C(s, a) + \alpha(t) \left[r_k + \gamma \max_{a'} Q_k(s'_k, a'_k) \right] \quad (14.11)$$

Where Q_k^C represents the Q-table containing states in which agent k will coordinate and Q_k contains the state-action values for its independent states. The joint state information is represented as s , whereas s_k and a_k are the local state information and action of agent k . So the update of Q_k^C uses the estimates of Q_k . This represents the one-step behaviour of the COORDINATE action and allows for a sparse representation of Q_k^C , since there is no direct dependency between the states in this joint Q-table.

Coordinating Q-Learning

Coordinating Q-Learning, or CQ-learning, learns in which states an agent should take the other agents into consideration (De Hauwere et al, 2010) and in which states it can act using primarily only its own state information. More precisely, the algorithm will identify states in which an agent should take other agents into account when choosing its preferred action.

The algorithm can be decomposed into three sections: detecting conflict situations, selecting actions and updating the Q-values which will now be explained in more detail:

1. Detecting conflict situations

Agents must identify in which states they experience the influence of at least one other agent. CQ-Learning needs a baseline for this, so agents are assumed to have learned a model about the expected payoffs for selecting an action in a particular state applying an individual policy. For example, in a gridworld this would mean that the agents have learned a policy to reach some goal, while being the only agent present in the environment. If agents are influencing each other, this will be reflected in the payoff the agents receive when they are acting together. CQ-learning uses a statistical test to detect if there are changes in the observed rewards for the selected state-action pair compared to the case where they were acting alone in the environment. Two situations can occur:

- a. The statistics allow to detect a change in the received immediate rewards. In this situation, the algorithm will mark this state, and search for the cause of this change by collecting new samples from the joint state space in order to identify the joint state-action pairs in which collisions occur. These state-action pairs are then marked as being *dangerous*, and the state space of the agent is augmented by adding this joint state information. State-action pairs that did not cause interactions are marked as being *safe*, i.e. the agent's actions in this state are independent from the states of other agents. So the algorithm will first attempt to detect changes in the rewards an agent receives, solely based on its own state, before trying to identify due to which other agents these changes occur.
- b. The statistics indicate that the rewards the agent receives are from the same distribution as if the agent was acting alone. Therefore, no special action is taken in this situation and the agent continues to act as if it was alone.

2. Selecting actions

If an agent selects an action, it will check if its current local state is a state in which a discrepancy has been detected previously (case 1.a, described above). If so, it will observe the global state information to determine if the state information of the other agents is the same as when the conflict was detected. If this is the case, it will condition its actions on this global state information, otherwise it can act independently, using only its own local state information. If its local state information has never caused a discrepancy (case 1.b, described above), it can act without taking the other agents into consideration.

3. Updating the Q-values

The updating the Q-values follows the same idea as the Learning of Coordination algorithm, described above. The Q-values for local states are used to bootstrap the Q-values of the states that were augmented.

The statistical test used in the algorithm is the Student t-test (Stevens, J.P., 1990). This test can determine whether the null hypothesis that the mean of two populations of samples are equal holds, against the alternative that they are not equal. In CQ-learning this test is first used to identify in which states the observed rewards are significantly different from the expected rewards based on single agent learning, and also to determine on which other agents' states these changes depend.

A formal description of this algorithm is given in Algorithm 25.

CQ-learning can also be used to generalise information from states in which coordination is necessary to obtain a state-independent representation of the coordination dependencies that exist between the agents (De Hauwere et al, 2010). This information can then be transferred to other, more complex, task environments (Vrancx et al, 2011). This principle of transfer learning improves the learning speed, since agents can purely focus on the core task of the problem at hand and use transferred experience for the coordination issues.

```

Initialise  $Q_k$  through single agent learning and  $Q_k^j$ ;
while true do
    if state  $s_k$  of Agent  $k$  is unmarked then
        Select  $a_k$  for Agent  $k$  from  $Q_k$ 
    else
        if the joint state information  $js$  is safe then
            Select  $a_k$  for Agent  $k$  from  $Q_k$ 
        else
            Select  $a_k$  for Agent  $k$  from  $Q_k^j$  based on the joint state information  $js$ 
        Sample  $\langle s_k, a_k, r_k \rangle$ 
        if t-test detects difference in observed rewards vs expected rewards for  $\langle s_k, a_k \rangle$  then
            mark  $s_k$ 
            for  $\forall$  other state information present in the joint state  $js$  do
                if t-test detects difference between independent state  $s_k$  and joint state  $js$  then
                    add  $js$  to  $Q_k^j$ 
                    mark  $js$  as dangerous
                else
                    mark  $js$  as safe
            if  $s_k$  is unmarked for Agent  $k$  or  $js$  is safe then
                No need to update  $Q_k(s_k)$ .
            else
                Update  $Q_k^j(js, a_k) \leftarrow (1 - \alpha_t)Q_k^j(js, a_k) + \alpha_t[r(js, a_k) + \gamma \max_a Q(s'_k, a)]$ 

```

Algorithm 25. CQ-Learning algorithm for agent k

This approach was later extended to detect sparse interactions that are only reflected in the reward signal, several timesteps in the future (De Hauwere et al, 2011). Examples of such situations are for instance if the order in which goods arrive in a warehouse are important.

14.5 Further Reading

Multi-agent reinforcement learning is a growing field of research, but quite some challenging research questions are still open. A lot of the work done in single-agent reinforcement learning, such as the work done in Bayesian RL, batch learning or transfer learning, has yet to find its way to the multi-agent learning community. General overviews of multi-agent systems are given by Weiss (Weiss, G., 1999), Wooldridge (Wooldridge, M., 2002) and more recently Shoham (Shoham, Y. and Leyton-Brown, K., 2009). For a thorough overview of the field of Game Theory the book by Gintis will be very useful (Gintis, H., 2000).

More focused on the domain of multi-agent reinforcement learning we recommend the paper by Buşoniu which gives an extensive overview of recent research and describes a representative selection of MARL algorithms in detail together with their strengths and weaknesses (Busoniu et al, 2008). Another track of multi-agent research considers systems where agents are not aware of the type or capabilities of the other agents in the system (Chalkiadakis and Boutilier, 2008).

An important issue in multi-agent reinforcement learning as well as in single agent reinforcement learning, is the fact that the reward signal can be delayed in time. This typically happens in systems which include queues, like for instance in network routing and job scheduling. The immediate feedback of taking an action can only be provided to the agent after the effect of the action becomes apparent, e.g. after the job is processed. In (Verbeek et al, 2005) a policy iteration approach based on learning automata is given, which is robust with respect to this type of delayed reward. In (Littman and Boyan, 1993) a value iteration based algorithm is described for routing in networks. An improved version of the algorithm is presented in (Steenhaut et al, 1997; Fakir, 2004). The improved version allows on one hand to use the feedback information in a more efficient way, and on the other hand it avoids instabilities that might occur due to careless exploration.

References

- Aumann, R.: Subjectivity and Correlation in Randomized Strategies. *Journal of Mathematical Economics* 1(1), 67–96 (1974)
- Bowling, M.: Convergence and No-Regret in Multiagent Learning. In: *Advances in Neural Information Processing Systems* 17 (NIPS), pp. 209–216 (2005)
- Bowling, M., Veloso, M.: Convergence of Gradient Dynamics with a Variable Learning Rate. In: *Proceedings of the Eighteenth International Conference on Machine Learning* (ICML), pp. 27–34 (2001)

- Bowling, M., Veloso, M.: Scalable Learning in Stochastic Games. In: AAAI Workshop on Game Theoretic and Decision Theoretic Agents (2002)
- Busoniu, L., Babuska, R., De Schutter, B.: A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 38(2), 156–172 (2008)
- Chalkiadakis, G., Boutilier, C.: Sequential Decision Making in Repeated Coalition Formation under Uncertainty. In: Parkes, P.M., Parsons (eds.) *Proceedings of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pp. 347–354 (2008)
- Claus, C., Boutilier, C.: The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In: *Proceedings of the National Conference on Artificial Intelligence*, pp. 746–752. John Wiley & Sons Ltd. (1998)
- De Hauwere, Y.M., Vrancx, P., Nowé, A.: Learning Multi-Agent State Space Representations. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems*, Toronto, Canada, pp. 715–722 (2010)
- De Hauwere, Y.M., Vrancx, P., Nowé, A.: Detecting and Solving Future Multi-Agent Interactions. In: *Proceedings of the AAMAS Workshop on Adaptive and Learning Agents*, Taipei, Taiwan, pp. 45–52 (2011)
- Dorigo, M., Stützle, T.: *Ant Colony Optimization*. Bradford Company, MA (2004)
- Fakir, M.: Resource Optimization Methods for Telecommunication Networks. PhD thesis, Department of Electronics and Informatics, Vrije Universiteit Brussel, Belgium (2004)
- Foster, D., Young, H.: Regret Testing: A Simple Payoff-based Procedure for Learning Nash Equilibrium. University of Pennsylvania and Johns Hopkins University, Mimeo (2003)
- Gillette, D.: Stochastic Games with Zero Stop Probabilities. *Ann. Math. Stud.* 39, 178–187 (1957)
- Gintis, H.: *Game Theory Evolving*. Princeton University Press (2000)
- Greenwald, A., Hall, K., Serrano, R.: Correlated Q-learning. In: *Proceedings of the Twentieth International Conference on Machine Learning*, pp. 242–249 (2003)
- Guestrin, C., Lagoudakis, M., Parr, R.: Coordinated Reinforcement Learning. In: *Proceedings of the 19th International Conference on Machine Learning*, pp. 227–234 (2002a)
- Guestrin, C., Venkataraman, S., Koller, D.: Context-Specific Multiagent Coordination and Planning with Factored MDPs. In: *18th National Conference on Artificial Intelligence*, pp. 253–259. American Association for Artificial Intelligence, Menlo Park (2002b)
- Hart, S., Mas-Colell, A.: A Reinforcement Procedure Leading to Correlated Equilibrium. *Economic Essays: A Festschrift for Werner Hildenbrand*, 181–200 (2001)
- Hu, J., Wellman, M.: Nash Q-learning for General-Sum Stochastic Games. *The Journal of Machine Learning Research* 4, 1039–1069 (2003)
- Kapetanakis, S., Kudenko, D.: Reinforcement Learning of Coordination in Cooperative Multi-Agent Systems. In: *Proceedings of the National Conference on Artificial Intelligence*, pp. 326–331. AAAI Press, MIT Press, Menlo Park, Cambridge (2002)
- Kapetanakis, S., Kudenko, D., Strens, M.: Learning to Coordinate Using Commitment Sequences in Cooperative Multiagent-Systems. In: *Proceedings of the Third Symposium on Adaptive Agents and Multi-agent Systems (AAMAS-2003)*, p. 2004 (2003)
- Kok, J., Vlassis, N.: Sparse Cooperative Q-learning. In: *Proceedings of the 21st International Conference on Machine Learning*. ACM, New York (2004a)
- Kok, J., Vlassis, N.: Sparse Tabular Multiagent Q-learning. In: *Proceedings of the 13th Benelux Conference on Machine Learning*, Benelearn (2004b)
- Kok, J., Vlassis, N.: Collaborative Multiagent Reinforcement Learning by Payoff Propagation. *Journal of Machine Learning Research* 7, 1789–1828 (2006)

- Kok, J., 't Hoen, P., Bakker, B., Vlassis, N.: Utile Coordination: Learning Interdependencies among Cooperative Agents. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG 2005), pp. 29–36 (2005)
- Kononen, V.: Asymmetric Multiagent Reinforcement Learning. In: IEEE/WIC International Conference on Intelligent Agent Technology (IAT 2003), pp. 336–342 (2003)
- Könönen, V.: Policy Gradient Method for Team Markov Games. In: Yang, Z.R., Yin, H., Everson, R.M. (eds.) IDEAL 2004. LNCS, vol. 3177, pp. 733–739. Springer, Heidelberg (2004)
- Leyton-Brown, K., Shoham, Y.: Essentials of Game Theory: A Concise Multidisciplinary Introduction. Synthesis Lectures on Artificial Intelligence and Machine Learning 2(1), 1–88 (2008)
- Littman, M.: Markov Games as a Framework for Multi-Agent Reinforcement Learning. In: Proceedings of the Eleventh International Conference on Machine Learning, pp. 157–163. Morgan Kaufmann (1994)
- Littman, M.: Friend-or-Foe Q-learning in General-Sum Games. In: Proceedings of the Eighteenth International Conference on Machine Learning, pp. 322–328. Morgan Kaufmann (2001a)
- Littman, M.: Value-function Reinforcement Learning in Markov Games. Cognitive Systems Research 2(1), 55–66 (2001b), <http://www.sciencedirect.com/science/article/B6W6C-430G1TK-4/2/822caf1574be32ae91adf15de90becc4>, doi:10.1016/S1389-0417(01)00015-8
- Littman, M., Boyan, J.: A Distributed Reinforcement Learning Scheme for Network Routing. In: Proceedings of the 1993 International Workshop on Applications of Neural Networks to Telecommunications, pp. 45–51. Erlbaum (1993)
- Mariano, C., Morales, E.: DQL: A New Updating Strategy for Reinforcement Learning Based on Q-Learning. In: Flach, P.A., De Raedt, L. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, pp. 324–335. Springer, Heidelberg (2001)
- Melo, F., Veloso, M.: Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems. In: Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems, pp. 773–780 (2009)
- Nash, J.: Equilibrium Points in n-Person Games. Proceedings of the National Academy of Sciences of the United States of America, 48–49 (1950)
- Peshkin, L., Kim, K., Meuleau, N., Kaelbling, L.: Learning to Cooperate via Policy Search. In: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, UAI 2000, pp. 489–496. Morgan Kaufmann Publishers Inc., San Francisco (2000), <http://portal.acm.org/citation.cfm?id=647234.719893>
- Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2003)
- Sastray, P., Phansalkar, V., Thathachar, M.: Decentralized Learning of Nash Equilibria in Multi-Person Stochastic Games with Incomplete Information. IEEE Transactions on Systems, Man and Cybernetics 24(5), 769–777 (1994)
- Shapley, L.: Stochastic Games. Proceedings of the National Academy of Sciences 39(10), 1095–1100 (1953)
- Shoham, Y., Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press (2009)
- Singh, S., Kearns, M., Mansour, Y.: Nash Convergence of Gradient Dynamics in General-Sum Games. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, pp. 541–548 (2000)
- Smith, J.: Evolution and the Theory of Games. Cambridge Univ. Press (1982)

- Sobel, M.: Noncooperative Stochastic Games. *The Annals of Mathematical Statistics* 42(6), 1930–1935 (1971)
- Spaan, M., Melo, F.: Interaction-Driven Markov Games for Decentralized Multiagent Planning under Uncertainty. In: Proceedings of the 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 525–532. International Foundation for Autonomous Agents and Multiagent Systems (2008)
- Steenhaut, K., Nowé, A., Fakir, M., Dirkx, E.: Towards a Hardware Implementation of Reinforcement Learning for Call Admission Control in Networks for Integrated Services. In: Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications, vol. 3, p. 63. Lawrence Erlbaum (1997)
- Stevens, J.P.: Intermediate Statistics: A Modern Approach. Lawrence Erlbaum (1990)
- Sutton, R., McAllester, D., Singh, S., Mansour, Y.: Policy Gradient Methods for Reinforcement Learning with Function Approximation. In: Advances in Neural Information Processing Systems, vol. 12(22) (2000)
- Tsitsiklis, J.: Asynchronous stochastic approximation and Q-learning. *Machine Learning* 16(3), 185–202 (1994)
- Tuyls, K., Nowé, A.: Evolutionary Game Theory and Multi-Agent Reinforcement Learning. *The Knowledge Engineering Review* 20(01), 63–90 (2005)
- Verbeeck, K.: Coordinated Exploration in Multi-Agent Reinforcement Learning. PhD thesis, Computational Modeling Lab, Vrije Universiteit Brussel, Belgium (2004)
- Verbeeck, K., Nowé, A., Tuyls, K.: Coordinated Exploration in Multi-Agent Reinforcement Learning: An Application to Loadbalancing. In: Proceedings of the 4th International Conference on Autonomous Agents and Multi-Agent Systems (2005)
- Vrancx, P., Tuyls, K., Westra, R.: Switching Dynamics of Multi-Agent Learning. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), vol. 1, pp. 307–313. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2008a), <http://portal.acm.org/citation.cfm?id=1402383.1402430>
- Vrancx, P., Verbeeck, K., Nowé, A.: Decentralized Learning in Markov Games. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 38(4), 976–981 (2008b)
- Vrancx, P., De Hauwere, Y.M., Nowé, A.: Transfer learning for Multi-Agent Coordination. In: Proceedings of the 3th International Conference on Agents and Artificial Intelligence, Rome, Italy, pp. 263–272 (2011)
- Weiss, G.: Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence. The MIT Press (1999)
- Wheeler Jr., R., Narendra, K.: Decentralized Learning in Finite Markov Chains. *IEEE Transactions on Automatic Control* 31(6), 519–526 (1986)
- Williams, R.: Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8(3), 229–256 (1992)
- Wooldridge, M.: An Introduction to Multi Agent Systems. John Wiley and Sons Ltd. (2002)
- Wunder, M., Littman, M., Babes, M.: Classes of Multiagent Q-learning Dynamics with epsilon-greedy Exploration. In: Proceedings of the 27th International Conference on Machine Learning, Haifa, Israel, pp. 1167–1174 (2010)
- Zinkevich, M.: Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In: Machine Learning International Conference, vol. 20(2), p. 928 (2003)
- Zinkevich, M., Greenwald, A., Littman, M.: Cyclic equilibria in Markov games. In: Advances in Neural Information Processing Systems, vol. 18, p. 1641 (2006)

Chapter 15

Decentralized POMDPs

Frans A. Oliehoek

Abstract. This chapter presents an overview of the decentralized POMDP (Dec-POMDP) framework. In a Dec-POMDP, a team of agents collaborates to maximize a global reward based on local information only. This means that agents do not observe a Markovian signal during execution and therefore the agents' individual policies map from histories to actions. Searching for an optimal joint policy is an extremely hard problem: it is NEXP-complete. This suggests, assuming $\text{NEXP} \neq \text{EXP}$, that any optimal solution method will require doubly exponential time in the worst case. This chapter focuses on planning for Dec-POMDPs over a finite horizon. It covers the forward heuristic search approach to solving Dec-POMDPs, as well as the backward dynamic programming approach. Also, it discusses how these relate to the optimal Q-value function of a Dec-POMDP. Finally, it provides pointers to other solution methods and further related topics.

15.1 Introduction

Previous chapters generalized decision making to multiple agents (Chapter 14) and to acting under state uncertainty as in POMDPs (Chapter 12). This chapter generalizes further by considering situations with both state uncertainty and multiple agents. In particular, it focuses on teams of *collaborative* agents: the agents share a single objective. Such settings can be formalized by the framework of *decentralized POMDPs (Dec-POMDPs)* (Bernstein et al, 2002) or the roughly equivalent *multi-agent team decision problem* (Pynadath and Tambe, 2002). The basic idea of this model is illustrated in Figure 15.1, which depicts the two-agent case. At each stage, the agents independently take an action. The environment undergoes a state transition and generates a reward depending on the state and the actions of both agents. Finally, each agent receives an individual observation of the new state.

Frans A. Oliehoek
CSAIL, Massachusetts Institute of Technology
e-mail: fao@csail.mit.edu

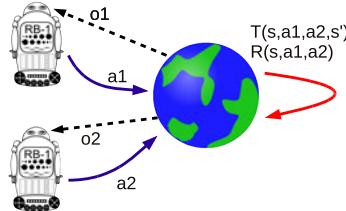


Fig. 15.1 Execution of a decentralized POMDP

This framework allows modeling important real-world tasks for which the models in the previous chapters do not suffice. An example of such a task is load balancing among queues (Cogill et al, 2004). Each agent represents a processing unit with a queue that has to decide whether to accept new jobs or pass them to another queue, based only on the local observations of its own queue size and that of immediate neighbors. Another important application area for Dec-POMDPs is communication networks. For instance, consider a packet routing task in which agents are routers that have to decide at each time step to which neighbor to send each packet in order to minimize the average transfer time of packets (Peshkin, 2001). An application domain that receives much attention in the Dec-POMDP community is that of sensor networks (Nair et al, 2005; Varakantham et al, 2007; Kumar and Zilberstein, 2009). Other areas of interests are teams of robotic agents (Becker et al, 2004b; Emery-Montemerlo et al, 2005; Seuken and Zilberstein, 2007a) and crisis management (Nair et al, 2003a,b; Paquet et al, 2005).

Most research on multi-agent systems under partial observability is relatively recent and has focused almost exclusively on planning—settings where the model of the environment is given—rather than the full reinforcement learning (RL) setting. This chapter also focuses exclusively on planning. Some pointers to RL approaches are given at the end of the chapter.

A common assumption is that planning takes place in an *off-line* phase, after which the plans are executed in an *on-line* phase. This on-line phase is completely *decentralized* as shown in Figure 15.1: each agent receives its individual part of the joint policy found in the planning phase¹ and its individual history of actions and observations. The off-line planning phase, however, is *centralized*. We assume a single computer that computes the joint plan and subsequently distributes it to the agents (who then merely execute the plan on-line).²

¹ In some cases it is assumed that the agents are given the joint policy. This enables the computation of a *joint belief* from broadcast local observations (see Section 15.5.4).

² Alternatively, each agent runs the same planning algorithm in parallel.

15.2 The Decentralized POMDP Framework

In this section we more formally introduce the Dec-POMDP model. We start by giving a mathematical definition of its components.

Definition 15.1 (Dec-POMDP). A *decentralized partially observable Markov decision process* is defined as a tuple $\langle \mathcal{D}, S, A, T, R, O, O, h, I \rangle$, where

- $\mathcal{D} = \{1, \dots, n\}$ is the set of n agents.
- S is a finite set of states s in which the environment can be.
- A is the finite set of joint actions.
- T is the transition probability function.
- R is the immediate reward function.
- O is the finite set of joint observations.
- O is the observation probability function.
- h is the horizon of the problem.
- $I \in \mathcal{P}(S)$, is the initial state distribution at stage $t = 0$.

The Dec-POMDP model extends single-agent POMDP models by considering *joint* actions and observations. In particular, $A = \times_{i \in \mathcal{D}} A^i$ is the set of *joint actions*. Here, A^i is the set of actions available to agent i , which can be different for each agent. Every time step, one joint action $a = \langle a^1, \dots, a^n \rangle$ is taken. How this joint action influences the environment is described by the transition function T , which specifies $P(s'|s, a)$. In a Dec-POMDP, agents only know their own individual action; they do not observe each other's actions. Similar to the set of joint actions, $O = \times_{i \in \mathcal{D}} O^i$ is the set of joint observations, where O^i is a set of observations available to agent i . Every time step the environment emits one joint observation $o = \langle o^1, \dots, o^n \rangle$ from which each agent i only observes its own component o^i . The observation function O specifies the probabilities $P(o|a, s')$ of joint observations. Figure 15.2 further illustrates the dynamics of the Dec-POMDP model.

During execution, the agents are assumed to act based on their individual observations only and no additional communication is assumed. This does not mean that Dec-POMDPs cannot model settings which concern communication. For instance, if one agent has an action “mark blackboard” and the other agent has an observation “marked blackboard”, the agents have a mechanism of communication through the state of the environment. However, rather than making this communication explicit, we say that the Dec-POMDP can model communication implicitly through the actions, states and observations. This means that in a Dec-POMDP, communication has no special semantics. Section 15.5.4 further elaborates on communication in Dec-POMDPs.

This chapter focuses on planning over a finite horizon, for which the (undiscounted) *expected cumulative reward* is the commonly used optimality criterion. The planning problem thus amounts to finding a tuple of policies, called a *joint policy* that maximizes the expected cumulative reward.

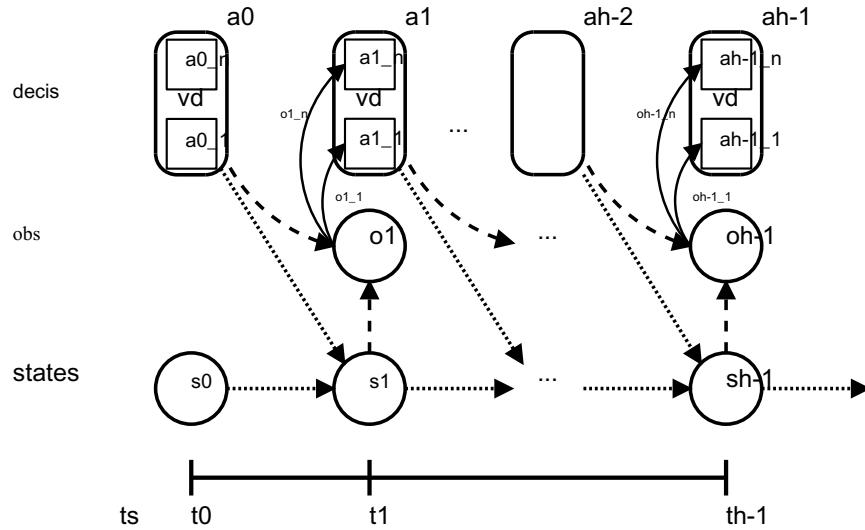


Fig. 15.2 A more detailed illustration of the dynamics of a Dec-POMDP. At every stage the environment is in a particular state. This state emits a joint observation according to the observation model (dashed arrows) from which each agent observes its individual component (indicated by solid arrows). Then each agent selects an action, together forming the joint action, which leads to a state transition according to the transition model (dotted arrows).

We will consider the decentralized tiger (Dec-Tiger) problem—a frequently used Dec-POMDP benchmark introduced by Nair et al (2003c)—as an example. It concerns two agents that are standing in a hallway with two doors. Behind one of the doors is a tiger, behind the other a treasure. Therefore there are two states: the tiger is behind the left door (s_l) or behind the right door (s_r). Both agents have 3 actions at their disposal: open the left door (a_{OL}), open the right door (a_{OR}) and listen (a_{Li}). They cannot observe each other's actions. In fact, they can only receive 2 observations: either they hear the tiger make a sound from behind the left (o_{HL}) or right (o_{HR}) door.

At $t = 0$ the state is s_l or s_r with probability 0.5. As long as no agent opens a door, the state does not change. When a door is opened, the state resets to s_l or s_r with probability 0.5. The observation probabilities are independent and identical for both agents. For instance, when the state is s_l and both perform action a_{Li} , each agent has a 85% chance of observing o_{HL} , and the probability that both hear the tiger behind the left door is $0.85 \times 0.85 = 0.72$. When one of the agents opens the door to the treasure they receive a positive reward (+9), while they receive a penalty for opening the wrong door (-101). When opening the wrong door jointly, the penalty is less severe (-50). Opening the correct door jointly leads to a higher reward (+20). The full transition, observation and reward models are listed by Nair et al (2003c).

Note that, when the wrong door is opened by one or both agents, they are attacked by the tiger and receive a penalty. However, neither of the agents observe this attack nor the penalty (remember, the only possible observations are o_{HL} and o_{HR}) and the episode continues. Intuitively, an optimal joint policy for Dec-Tiger should specify that the agents listen until they are certain enough to open one of the doors. At the same time, the policy should be ‘as coordinated’ as possible, i.e., maximize the probability of acting jointly.

15.3 Histories and Policies

In an MDP, the agent uses a policy that maps states to actions. In selecting its action, it can ignore the history because of the Markov property. In a POMDP, the agent can no longer observe the state, but it can compute a belief b that summarizes the history; it is also a Markovian signal. In a Dec-POMDP, however, during execution each agent will only have access to its *individual* actions and observations and there is no method known to summarize this individual history. It is not possible to maintain and update an individual belief in the same way as in a POMDP, because the transition and observation function are specified in terms of joint actions and observations.³

This means that in a Dec-POMDP *the agents do not have access to a Markovian signal during execution*. The consequence of this is that planning for Dec-POMDPs involves searching the space of tuples of individual Dec-POMDP policies that map full-length individual histories to actions. We will see later that this also means that solving Dec-POMDPs is even harder than solving POMDPs.

15.3.1 Histories

First, we define histories of observations, actions and both.

Definition 15.2 (Action-observation history). The *action-observation history (AOH)* for agent i , $\bar{\theta}^i$, is the sequence of actions taken by and observations received by agent i . At a specific time step t , this is

$$\bar{\theta}_t^i = (a_0^i, o_1^i, \dots, a_{t-1}^i, o_t^i).$$

The *joint action-observation history* $\bar{\theta}_t = \langle \bar{\theta}_t^1, \dots, \bar{\theta}_t^n \rangle$ specifies the AOH for all agents. Agent i ’s set of possible AOHs at time t is $\bar{\Theta}_t^i$. The set of AOHs possible for

³ Different forms of beliefs for Dec-POMDP-like settings have been considered Nair et al (2003c); Hansen et al (2004); Oliehoek et al (2009); Zettlemoyer et al (2009). These are not specified over only states, but also specify probabilities over histories/policies/types/beliefs of the other agents. The key point is that from an individual agent’s perspective just knowing a probability distribution over states is insufficient; it also needs to predict what actions the other agents will take.

all stages for agent i is $\bar{\Theta}^i$ and $\bar{\theta}^i$ denotes an AOH from this set.⁴ Finally the set of all possible *joint* AOHs $\bar{\Theta}$ is denoted $\bar{\Theta}$. At $t = 0$, the (joint) AOH is empty $\bar{\Theta}_0 = ()$.

Definition 15.3 (Observation history). The *observation history (OH)* for agent i , \bar{o}^i , is defined as the sequence of observations an agent has received. At a specific time step t , this is:

$$\bar{o}_t^i = (o_1^i, \dots, o_t^i).$$

The *joint observation history*, is the OH for all agents: $\bar{o}_t = \langle \bar{o}_t^1, \dots, \bar{o}_t^n \rangle$. The set of observation histories for agent i at time t is denoted \bar{O}_t^i . Similar to the notation for action-observation histories, we also use $\bar{o}^i \in \bar{O}^i$ and $\bar{o} \in \bar{O}$.

Definition 15.4 (Action history). The *action history (AH)* for agent i , \bar{a}^i , is the sequence of actions an agent has performed:

$$\bar{a}_t^i = (a_0^i, a_1^i, \dots, a_{t-1}^i).$$

Notation for joint action histories and sets are analogous to those for observation histories. Finally, note that a (joint) AOH consists of a (joint) action- and a (joint) observation history: $\bar{\theta}_t = \langle \bar{o}_t, \bar{a}_t \rangle$.

15.3.2 Policies

A policy π^i for an agent i maps from histories to actions. In the general case, these histories are AOHs, since they contain all information an agent has. The number of AOHs grows exponentially with the horizon of the problem: At time step t , there are $(|A^i| |s| |O^i|)^t$ possible AOHs for agent i . A policy π^i assigns an action to each of these histories. As a result, the number of possible policies π^i is doubly exponential in the horizon.

It is possible to reduce the number of policies under consideration by realizing that many policies of the form considered above specify the same behavior. This is illustrated by the left side of Figure 15.3: under a deterministic policy only a subset of possible action-observation histories can be reached. Policies that only differ with respect to an AOH that can never be reached, manifest the same behavior. The consequence is that in order to specify a deterministic policy, the observation history suffices: when an agent selects actions deterministically, it will be able to infer what action it took from only the observation history. This means that a deterministic policy can conveniently be represented as a tree, as illustrated by the right side of Figure 15.3.

Definition 15.5. A *deterministic policy* π^i for agent i is a mapping from observation histories to actions, $\pi^i : \bar{O}^i \rightarrow A^i$.

⁴ In a particular Dec-POMDP, it may be the case that not all of these histories can actually be realized, because of the probabilities specified by the transition and observation model.

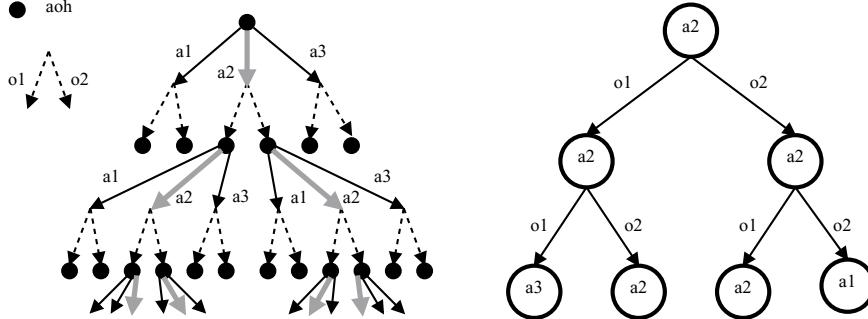


Fig. 15.3 A deterministic policy can be represented as a tree. Left: a tree of action-observation histories $\bar{\theta}^i$ for one of the agents from the Dec-Tiger problem. A deterministic policy π^i is highlighted. Clearly shown is that π^i only reaches a subset of histories $\bar{\theta}^i$. ($\bar{\theta}^i$ that are not reached are not further expanded.) Right: The same policy can be shown in a simplified policy tree. When both agents execute this policy in the $h = 3$ Dec-Tiger problem, the joint policy is optimal.

For a deterministic policy, $\pi^i(\bar{\theta}^i)$ denotes the action that it specifies for the observation history contained in $\bar{\theta}^i$. For instance, let $\bar{\theta}^i = \langle \bar{o}^i, \bar{a}^i \rangle$, then $\pi^i(\bar{\theta}^i) \triangleq \pi^i(\bar{o}^i)$. We use $\pi = \langle \pi^1, \dots, \pi^n \rangle$ to denote a *joint policy*. We say that a deterministic joint policy is an *induced mapping* from joint observation histories to joint actions $\pi : \bar{O} \rightarrow A$. That is, the mapping is induced by individual policies π^i that make up the joint policy. Note, however, that only a subset of possible mappings $f : \bar{O} \rightarrow A$ correspond to valid joint policies: when f does not specify the same individual action for each \bar{o}^i of each agent i , it will not be possible to execute f in a decentralized manner. That is, such a policy is *centralized*: it would describe that an agent should base its choice of action on the *joint* history. However, during execution it will only be able to observe its *individual* history, not the joint history.

Agents can also execute stochastic policies, but we restrict our attention to deterministic policies without sacrificing optimality, since a finite-horizon Dec-POMDP has at least one optimal pure joint policy (Oliehoek et al, 2008b).

15.3.3 Structure in Policies

Policies specify actions for all stages of the Dec-POMDP. A common way to represent the temporal structure in a policy is to split it into *decision rules* δ^i that specify the policy for each stage. An individual policy is then represented as a sequence of decision rules $\pi^i = (\delta_0^i, \dots, \delta_{h-1}^i)$. In case of a deterministic policy, the form of the decision rule for stage t is a mapping from length- t observation histories to actions $\delta_t^i : \bar{O}_t^i \rightarrow A^i$. In the more general case its domain is the set of AOHs $\delta_t^i : \bar{\Theta}_t^i \rightarrow A^i$. A joint decision rule $\delta_t = \langle \delta_t^1, \dots, \delta_t^n \rangle$ specifies a decision rule for each agent.

We will also consider policies that are partially specified with respect to time. Formally, $\varphi_t = (\delta_0, \dots, \delta_{t-1})$ denotes the *past joint policy* at stage t , which is a partial joint policy specified for stages $0, \dots, t-1$. By appending a joint decision rule for stage t , we can ‘grow’ such a past joint policy.

Definition 15.6 (Policy concatenation). We write

$$\varphi_{t+1} = (\delta_0, \dots, \delta_{t-1}, \delta_t) = \langle \varphi_t \circ \delta_t \rangle \quad (15.1)$$

to denote policy concatenation.

A *future policy* ψ_t^i of agent i specifies all the future behavior relative to stage t . That is, $\psi_t^i = (\delta_{t+1}^i, \dots, \delta_{h-1}^i)$. We also consider future joint policies $\psi_t = (\delta_{t+1}, \dots, \delta_{h-1})$. The structure of a policy π^i can be represented as

$$\pi^i = (\underbrace{\delta_0^i, \delta_1^i, \dots, \delta_{t-1}^i}_{\varphi_t^i}, \delta_t^i, \underbrace{\delta_{t+1}^i, \dots, \delta_{h-1}^i}_{\psi_t^i}) \quad (15.2)$$

and similarly for joint policies.

Since policies can be represented as trees (remember Figure 15.3), a different way to decompose them is by considering sub-trees. Define the *time-to-go* τ at stage t as

$$\tau = h - t.$$

Now $q_{\tau=k}^i$ denotes a k -steps-to-go *sub-tree policy* for agent i . That is, $q_{\tau=k}^i$ is a policy tree that has the same form as a full policy for the horizon- k problem. Within the original horizon- h problem $q_{\tau=k}^i$ is a candidate for execution starting at stage $t = h - k$. The set of k -steps-to-go sub-tree policies for agent i is denoted $Q_{\tau=k}^i$. A joint sub-tree policy $q_{\tau=k} \in Q_{\tau=k}$ specifies a sub-tree policy for each agent.

Figure 15.4 shows the different structures in a policy for a fictitious Dec-POMDP with $h = 3$. It represents decision rules by dotted ellipses. It also shows a past policy φ_2^i and illustrates how policy concatenation $\langle \varphi_2^i \circ \delta_2^i \rangle = \pi^i$ forms the full policy. This full policy also corresponds to a 3-steps-to-go sub-tree policy $q_{\tau=3}^i$; two of the sub-tree policies are indicated using dashed ellipses.

Definition 15.7 (Policy consumption). Providing a length- k (joint) sub-tree policy $q_{\tau=k}$ with a sequence of $l < k$ (joint) observations *consumes* a part of $q_{\tau=k}$ leading to a (joint) sub-tree policy which is a sub-tree of $q_{\tau=k}$. In particular, consumption \Downarrow by a single joint observation o is written as

$$q_{\tau=k-1} = q_{\tau=k} \Downarrow_o. \quad (15.3)$$

For instance, in Figure 15.4, $q_{\tau=1}^i = q_{\tau=2}^i \Downarrow_o$.

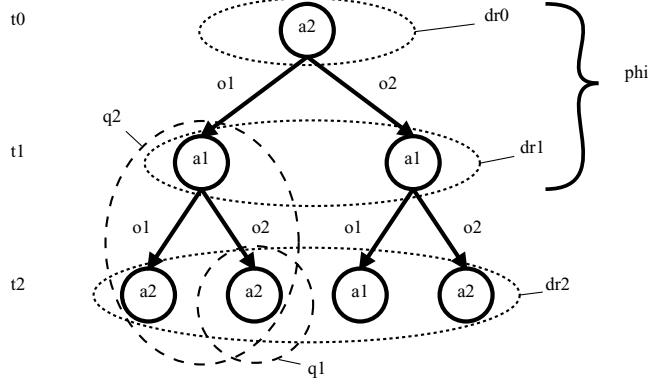


Fig. 15.4 Structure of a policy for an agent with actions $\{a, \check{a}\}$ and observations $\{o, \check{o}\}$. A policy π^i can be divided into decision rules δ^i or sub-tree policies q^i .

15.3.4 The Quality of Joint Policies

Joint policies differ in how much reward they can expect to accumulate, which will serve as a criterion of their quality. Formally, we consider the expected cumulative reward of a joint policy, also referred to as its *value*.

Definition 15.8. The *value* $V(\pi)$ of a joint policy π is defined as

$$V(\pi) \triangleq E\left[\sum_{t=0}^{h-1} R(s^t, a_t) \mid I, \pi\right],$$

where the expectation is over states and observations.

This expectation can be computed using a recursive formulation. For the last stage $t = h - 1$, the value is given simply by the immediate reward

$$V^\pi(s_{h-1}, \bar{o}_{h-1}) = R(s_{h-1}, \pi(\bar{o}_{h-1})).$$

For all other stages, the expected value is given by:

$$V^\pi(s_t, \bar{o}_t) = R(s_t, \pi(\bar{o}_t)) + \sum_{s_{t+1} \in S} \sum_{o \in O} P(s_{t+1}, o | s_t, \pi(\bar{o}_t)) V^\pi(s_{t+1}, \bar{o}_{t+1}). \quad (15.4)$$

Here, the probability is simply the product of the transition and observation probabilities $P(s', o | s, a) = P(o | a, s') P(s' | s, a)$. In essence, fixing the joint policy transforms the Dec-POMDP to a Markov chain with states (s_t, \bar{o}_t) . Evaluating this equation via dynamic programming will result in the value for all (s_0, \bar{o}_0) -pairs. The value $V(\pi)$ is then given by weighting these pairs according to the initial state distribution I . Note that given a fixed joint policy π , a history \bar{o}_t actually induces a joint sub-tree

policy. As such, it is possible to rewrite (15.4) in terms of sub-tree policies. Executing $q_{\tau=k}$ over the last k stages, starting from a state s at stage $t = h - k$ will achieve:

$$V(s_t, q_{\tau=k}) = R(s_t, a_t) + \sum_{s_{t+1} \in S} \sum_{o \in O} P(s_{t+1}, o | s_t, a_t) V(s_{t+1}, q_{\tau=k} \Downarrow_o) \quad (15.5)$$

where a_t is the joint action specified by (the root of) $q_{\tau=k}$.

Finally, as is apparent from the above equations, the probabilities of states and histories are important in many computations. The following equation recursively specifies the probabilities of states and joint AOHs under a (potentially stochastic) past joint policy:

$$\begin{aligned} P(s_t, \bar{\theta}_t | I, \varphi_t) &= \sum_{s_{t-1} \in S} \sum_{a_{t-1} \in A} P(s_t, o_t | s_{t-1}, a_{t-1}) P(a_{t-1} | \bar{\theta}_{t-1}, \varphi_t) \\ &\quad P(s_{t-1}, \bar{\theta}_{t-1} | I, \varphi_t). \end{aligned} \quad (15.6)$$

15.4 Solution of Finite-Horizon Dec-POMDPs

This section gives an overview of methods proposed for finding exact and approximate solutions for finite-horizon Dec-POMDPs. For the infinite-horizon problem, which is significantly different, some pointers are provided in Section 15.5.

15.4.1 Brute Force Search and Dec-POMDP Complexity

Because there exists an optimal deterministic joint policy for a finite-horizon Dec-POMDP, it is possible to enumerate all joint policies, evaluate them as described in Section 15.3.4 and choose the best one. However, the number of such joint policies is

$$O\left(|A^\dagger|^{\frac{n(|O^\dagger|^{h-1})}{|O^\dagger|-1}}\right),$$

where $|A^\dagger|$ and $|O^\dagger|$ denote the largest individual action and observation sets. The cost of evaluating each joint policy is $O(|S|s|O^\dagger|^{nh})$. It is clear that this approach therefore is only suitable for very small problems. This analysis provides some intuition about how hard the problem is. This intuition is supported by the complexity result due to Bernstein et al (2002).

Theorem 15.1 (Dec-POMDP complexity). *The problem of finding the optimal solution for a finite-horizon Dec-POMDP with $n \geq 2$ is NEXP-complete.*

NEXP is the class of problems that takes non-deterministic exponential time. Non-deterministic means that, similar to NP, it requires generating a guess about the

solution in a non-deterministic way. Exponential time means that verifying whether the guess is a solution takes exponential time. In practice this means that (assuming $\text{NEXP} \neq \text{EXP}$) solving a Dec-POMDP takes doubly exponential time in the worst case. Moreover, Dec-POMDPs cannot be approximated efficiently: Rabinovich et al (2003) showed that even finding an ε -approximate solution is NEXP-complete.

15.4.2 Alternating Maximization

Joint Equilibrium based Search for Policies (JESP) (Nair et al, 2003c) is a method that is guaranteed to find a locally optimal joint policy, more specifically, a *Nash equilibrium*: a tuple of policies such that for each agent i its policy π^i is a best response for the policies employed by the other agents π^{-i} . It relies on a process called *alternating maximization*. This is a procedure that computes a policy π^i for an agent i that maximizes the joint reward, while keeping the policies of the other agents fixed. Next, another agent is chosen to maximize the joint reward by finding its best response. This process is repeated until the joint policy converges to a Nash equilibrium, which is a local optimum. This process is also referred to as *hill-climbing* or *coordinate ascent*. Note that the local optimum reached can be arbitrarily bad. For instance, if agent 1 opens the left (a_{OL}) door right away in the Dec-Tiger problem, the best response for agent 2 is to also select a_{OL} . To reduce the impact of such bad local optima, JESP can use random restarts.

JESP uses a dynamic programming approach to compute the best-response policy for a selected agent i . In essence, fixing π^{-i} allows for a reformulation of the problem as an augmented POMDP. In this augmented POMDP a state $\check{s} = \langle s, \bar{o}^{-i} \rangle$ consists of a nominal state s and the observation histories of the other agents \bar{o}^{-i} . Given the fixed deterministic policies of other agents π^{-i} , such an augmented state \check{s} is Markovian and all transition and observation probabilities can be derived from π^{-i} and the transition and observation model of the original Dec-POMDP.

15.4.3 Optimal Value Functions for Dec-POMDPs

This section describes an approach more in line with methods for single agent MDPs and POMDPs: we identify an optimal value function Q^* that can be used to derive an optimal policy. Even though computation of Q^* itself is intractable, the insight it provides is valuable. In particular, it has a clear relation with the two dominant approaches to solving Dec-POMDPs: the forward and the backward approach which will be explained in the following subsections.

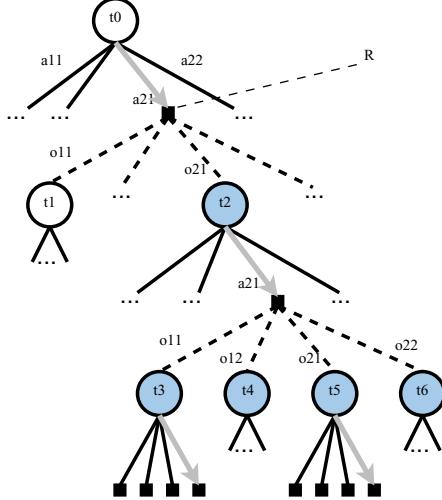


Fig. 15.5 Tree of joint AOHs $\bar{\theta}$ for a fictitious 2-agent Dec-POMDP with actions $\{\{a_1, \check{a}_1\}, \{a_2, \check{a}_2\}\}$ and observations $\{\{o_1, \check{o}_1\}, \{o_2, \check{o}_2\}\}$. Given I , the AOHs induce a ‘joint belief’ $b(s)$ over states. Solid lines represent joint actions and dashed lines joint observations. Due to the size of the tree it is only partially shown. Highlighted joint actions represent a joint policy. Given a joint sub-tree policy at a node (the action choices made in the sub-tree below it), the value is given by (15.8). However, action choices are not independent in different parts of the trees: e.g., the two nodes marked † have the same $\bar{\theta}^1$ and therefore should specify the same sub-tree policy for agent 1.

15.4.3.1 Selecting Sub-Tree Policies

Let us start by considering Figure 15.5, which illustrates a tree of joint AOHs. For a particular joint AOH (a node in Figure 15.5), we can try to determine which joint sub-tree policy $q_{\tau=k}$ is optimal. Recall that $V(s_t, q_{\tau=k})$ the value of $q_{\tau=k}$ starting from s_t is specified by (15.5). Also, let $b(s) \triangleq P(s|I, \bar{\theta}_t)$ be the *joint belief* corresponding to $\bar{\theta}_t$ which can be computed using Bayes’ rule in the same way as the POMDP belief update (see Chapter 12). Given an initial belief I and joint AOH $\bar{\theta}_t$, we can compute a value for each joint sub-tree policy $q_{\tau=k}$ that can be used from that node onward via

$$V(I, \bar{\theta}_t, q_{\tau=k}) = \sum_{s \in S} b(s) V(s, q_{\tau=k}). \quad (15.7)$$

Now, it is possible to rewrite (15.7) recursively:

$$V(I, \bar{\theta}_t, q_{\tau=k}) = R(\bar{\theta}_t, a_t) + \sum_o P(o|b, a) V(I, \bar{\theta}_{t+1}, q_{\tau=k} \parallel_o), \quad (15.8)$$

where the expected immediate reward is given by:

$$R(\bar{\theta}_t, a) = \sum_{s_t \in S} b(s_t) R(s_t, a). \quad (15.9)$$

Therefore one would hope that a dynamic programming approach would be possible, where, for each $\bar{\theta}_t$, one could choose the maximizing $q_{\tau=k}$. Unfortunately, running such a procedure on the entire tree is not possible because of the decentralized nature of a Dec-POMDP: it is not possible to choose maximizing joint sub-tree policies $q_{\tau=k}$ independently, since this could lead to a centralized joint policy.

The consequence is that, even though (15.8) can be used to compute the value for a $(\bar{\theta}_t, q_{\tau=k})$ -pair, it does not directly help to optimize the joint policy, because we cannot reason about parts of the joint AOH tree independently. Instead, one should decide what sub-tree policies to select by considering all $\bar{\theta}_t$ of an entire stage t *at the same time, assuming a past joint policy φ_t* . That is, when we assume we have computed $V(I, \bar{\theta}_t, q_{\tau=k})$ for all $\bar{\theta}_t$ and for all $q_{\tau=k}$, then we can compute a special form of joint decision rule $\Gamma_t = \langle \Gamma_t^i \rangle_{i \in \mathcal{D}}$ for stage t . Here, the individual decision rules map individual histories to individual sub-tree policies $\Gamma_t^i : \bar{\Theta}_t^i \rightarrow Q_{\tau=k}^i$. The optimal Γ_t satisfies:

$$\Gamma_t^* = \arg \max_{\Gamma_t} \sum_{\bar{\theta}_t \in \bar{\Theta}_t} P(\bar{\theta}_t | I, \varphi_t) V(I, \bar{\theta}_t, \Gamma_t(\bar{\theta}_t)), \quad (15.10)$$

where $\Gamma_t(\bar{\theta}_t) = \langle \Gamma_t^i(\bar{\theta}_t^i) \rangle_{i \in \mathcal{D}}$ denotes the joint sub-tree policy $q_{\tau=k}$ resulting from application of the individual decision rules and the probability is a marginal of (15.6).

This equation clearly illustrates that the optimal joint policy at a stage t of a Dec-POMDP depends on φ_t , the joint policy followed up to stage t . Moreover, there are additional complications that make (15.10) impractical to use:

1. It sums over joint AOHs, the number of which is exponential in both the number of agents and t .
2. It assumes computation of $V(I, \bar{\theta}_t, q_{\tau=k})$ for all $\bar{\theta}_t$, for all $q_{\tau=k}$.
3. The number of Γ_t to be evaluated is $O(|Q_{\tau=k}^\dagger|^{|\bar{\Theta}_t^\dagger|^n})$, where ‘ \dagger ’ denotes the largest set. $|Q_{\tau=k}^\dagger|$ is doubly exponential in k and $|\bar{\Theta}_t^\dagger|$ is exponential in t . Therefore the number of Γ_t is doubly exponential in $h = t + k$.

Note that by restricting our attention to deterministic φ_t it is possible to reformulate (15.10) as a summation over OHs, rather than AOHs (this involves adapting V to take φ_t as an argument). However, for such a reformulation, the same complications hold.

15.4.3.2 Selecting Optimal Decision Rules

This section shifts the focus back to regular decision rules δ^i —as introduced in Section 15.3.3—that map from OHs (or AOHs) to *actions*. We will specify a value function that quantifies the expected value of taking actions as specified by δ_t and

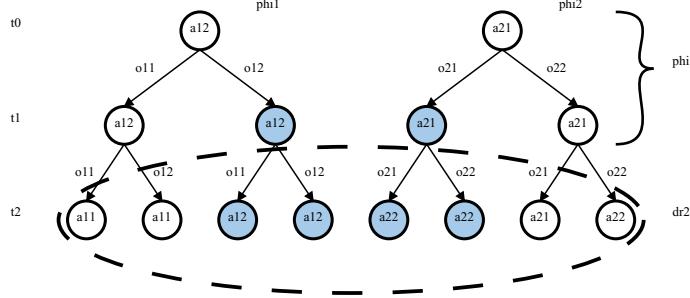


Fig. 15.6 Computation of Q^* . The dashed ellipse indicates the optimal decision rule δ_2^* for stage $t = 2$, given that $\varphi_2 = \langle \varphi_1 \circ \delta_1 \rangle$ is followed for the first two stages. The entries $Q^*(I, \bar{\theta}_1, \varphi_1, \delta_1)$ are computed by propagating relevant Q^* -values of the next stage. For instance, the Q^* -value under φ_2 for the highlighted joint history $\bar{\theta}_1 = \langle (\bar{a}_1, \dot{o}_1), (a_2, o_2) \rangle$ is computed by propagating the values of the four successor joint histories, as per (15.12).

continuing optimally afterward. That is, we replace the value of sub-trees in (15.10) by the optimal value of decision rules. The optimal value function for a finite-horizon Dec-POMDP is defined as follows.

Theorem 15.2 (Optimal Q^*). *The optimal Q -value function $Q^*(I, \varphi_t, \bar{\theta}_t, \delta_t)$ is a function of the initial state distribution and joint past policy, AOH and decision rule. For the last stage, it is given by*

$$Q^*(I, \varphi_{h-1}, \bar{\theta}_{h-1}, \delta_{h-1}) = R(\bar{\theta}_{h-1}, \delta_{h-1}(\bar{\theta}_{h-1})), \quad (15.11)$$

as defined by (15.9), and, for all $0 \leq t < h - 1$, by

$$Q^*(I, \varphi_t, \bar{\theta}_t, \delta_t) = R(\bar{\theta}_t, \delta_t(\bar{\theta}_t)) + \sum_o P(o | \bar{\theta}_t, \delta_t(\bar{\theta}_t)) Q^*(I, \varphi_{t+1}, \bar{\theta}_{t+1}, \delta_{t+1}^*), \quad (15.12)$$

with $\varphi_{t+1} = \langle \varphi_t \circ \delta_t \rangle$, $\bar{\theta}_{t+1} = (\bar{\theta}_t, \delta_t(\bar{\theta}_t), o)$ and

$$\delta_{t+1}^* = \arg \max_{\delta_{t+1}} \sum_{\bar{\theta}_{t+1} \in \bar{\Theta}_{t+1}} P(\bar{\theta}_{t+1} | I, \varphi_{t+1}) Q^*(I, \varphi_{t+1}, \bar{\theta}_{t+1}, \delta_{t+1}). \quad (15.13)$$

Proof. Because of (15.11), application of (15.13) for the last stage will maximize the expected reward and thus is optimal. Equation (15.12) propagates these optimal values to the preceding stage. Optimality for all stages follows by induction.

Note that φ_t is necessary in order to compute δ_{t+1}^* , the optimal joint decision rule at the next stage, because (15.13) requires φ_{t+1} and thus φ_t .

The above equations constitute a dynamic program. When assuming that only deterministic joint past policies φ can be used, the dynamic program can be evaluated from the end ($t = h - 1$) to the beginning ($t = 0$). Figure 15.6 illustrates the

computation of Q^* . When arriving at stage 0, the past joint policy is empty $\varphi_0 = ()$ and joint decision rules are simply joint actions, thus it is possible to select

$$\delta_0^* = \arg \max_{\delta_0} Q^*(I, \varphi_0, \bar{\theta}_0, \delta_0) = \arg \max_a Q^*(I, (), (), a).$$

Then given $\varphi_1 = \delta_0^*$ we can determine δ_1^* using (15.13), etc.⁵ This procedure, we refer to as *forward-sweep policy computation (FSPC)* using Q^* . The principle of FSPC is that a new decision rule is selected given the past joint policy found so far and is illustrated in Figure 15.7a.

Unfortunately, computing Q^* itself is intractable, since it means evaluating the dynamic program of Theorem 15.2 for all past joint policies. In particular, (15.11) will need to be evaluated for all $(\varphi_{h-1}, \delta_{h-1})$ and these pairs have a direct correspondence to all joint policies: $\pi = \langle \varphi_{h-1} \circ \delta_{h-1} \rangle$. Therefore, the time needed to evaluate this DP is doubly exponential in h . This means that the practical value of Q^* is limited.

The identified Q^* has a form quite different from Q-value functions encountered in MDPs and POMDPs. We still use the symbol ‘Q’, because δ_t can be seen as an action on the meta-level of the planning process. In this process (I, φ_t) can be interpreted as the state and we can define V and Q with their usual interpretations. In particular, it is possible to write

$$V^*(I, \varphi_t) = \max_{\delta_t} Q^*(I, \varphi_t, \delta_t) \quad (15.14)$$

where Q^* is defined as

$$Q^*(I, \varphi_t, \delta_t) = \sum_{\bar{\theta}_t} P(\bar{\theta}_t | I, \varphi_t) Q^*(I, \varphi_t, \bar{\theta}_t, \delta_t).$$

By expanding this definition of Q^* using (15.12), one can verify that it indeed has the regular interpretation of the expected immediate reward induced by first taking ‘action’ δ_t plus the cumulative reward of continuing optimally afterward (Oliehoek, 2010).

15.4.4 Forward Approach: Heuristic Search

The previous section explained that once Q^* is computed, it is possible to extract π^* by performing forward-sweep policy computation: repeatedly applying (15.13) for consecutive stages $t = 0, 1, \dots, h - 1$. When processing stage t , stages $0 \dots t - 1$ have been processed already. Therefore a past joint policy $\varphi_t = (\delta_0, \dots, \delta_{t-1})$ is

⁵ Note that performing the maximization in (15.13) has already been done and can be cached.

available and the probability $P(\bar{\theta}_t | I, \varphi_t)$ is defined. Unfortunately, computing Q^* itself is intractable. One idea to overcome this problem is to use an approximation \hat{Q} that is easier to compute. We refer to this as *the forward approach to Dec-POMDPs*.

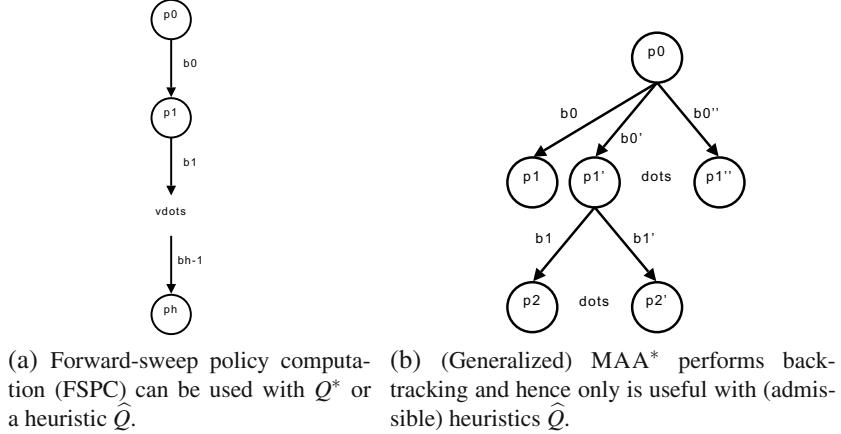


Fig. 15.7 Forward approach to Dec-POMDPs

15.4.4.1 Dec-POMDPs as Series of Bayesian Games

A straightforward approach is to try and apply forward-sweep policy computation using a heuristic Q-value function \hat{Q} . This is essentially what the method introduced by Emery-Montemerlo et al (2004) does. It represents a Dec-POMDP as a series of *collaborative Bayesian games (CBGs)*, one for each stage t , with an approximate payoff function $\hat{Q}(\bar{\theta}_t, a)$. A Bayesian game (BG) (Osborne and Rubinstein, 1994) is an extension of a strategic form game in which the agents have private information. A CBG is a BG with identical payoffs. By solving CBGs for consecutive stages it is possible to find an approximate solution. This is forward-sweep policy computation (with \hat{Q}).

In a Dec-POMDP, the crucial difficulty in making a decision at some stage t is that the agents lack a common signal on which to condition their actions. They must instead base their actions on their individual histories. Given I and φ_t , this situation can be modeled as a CBG. Such a CBG $B(I, \varphi_t)$ consists of:

- the set of agents,
- their joint actions A ,
- the set of their joint AOHs $\bar{\Theta}_t$,
- a probability distribution over them $P(\bar{\theta}_t | I, \varphi_t)$, and
- a payoff function $\hat{Q}(\bar{\theta}_t, a)$.

In the CBG agents use policies that map from their individual AOHs to actions. That is, a policy of an agent i for a CBG corresponds to a decision rule δ_t^i for the Dec-POMDP. The solution of the CBG is the joint decision rule δ_t that maximizes the expected payoff with respect to \widehat{Q} :

$$\widehat{\delta}_t^* = \arg \max_{\delta_t} \sum_{\bar{\theta}_t \in \bar{\Theta}_t} P(\bar{\theta}_t | I, \varphi_t) \widehat{Q}(\bar{\theta}_t, \delta_t(\bar{\theta}_t)). \quad (15.15)$$

Again, if φ_t is deterministic, the probability of $\bar{\theta}_t = \langle \bar{a}_t, \bar{o}_t \rangle$ is non-zero for exactly one \bar{a}_t , which means that attention can be restricted to OHs and decision rules that map from OHs to actions.

15.4.4.2 Heuristic Q-Value Functions

While the CBG for a stage is fully specified given I, φ_t and \widehat{Q} , it is not obvious how to choose \widehat{Q} . Here we discuss this issue.

Note that, for the last stage $t = h - 1$, $\widehat{\delta}_t^*$ has a close relation⁶ with the optimal decision rule selected by (15.13): if for the last stage the heuristic specifies the immediate reward $\widehat{Q}(\bar{\theta}_t, a) = R(\bar{\theta}_t, a)$, both will select the same actions. That is, in this case $\widehat{\delta}_t^* = \delta_t^*$.

While for other stages it is not possible to specify such a strong correspondence, note that FSPC via CBGs is not sub-optimal per se: It is possible to compute a value function of the form $Q^\pi(\bar{\theta}_t, a)$ for any π . Doing this for a π^* yields Q^{π^*} and when using the latter as the payoff functions for the CBGs, FSPC is exact (Oliehoek et al, 2008b).⁷

However, the practical value of this insight is limited, since it requires knowing an optimal policy to start with. In practice, research has considered using an approximate value function. For instance, it is possible to compute the value function $Q_M(s, a)$ of the ‘underlying MDP’: the MDP with the same transition and reward function as the Dec-POMDP (Emery-Montemerlo et al, 2004; Szer et al, 2005). This can be used to compute $\widehat{Q}(\bar{\theta}_t, a) = \sum_s b(s) Q_M(s, a)$, which can be used as the payoff function for the CBGs. This is called Q_{MDP} . Similarly, it is possible to use the value function of the ‘underlying POMDP’ (Q_{POMDP}) (Roth et al, 2005b; Szer et al, 2005), or the value function of the problem with 1-step delayed communication (Q_{BG}) (Oliehoek and Vlassis, 2007).

A problem in FSPC is that (15.15) still maximizes over δ_t that map from histories to actions; the number of such δ_t is doubly exponential in t . There are two main approaches to gain leverage. First, the maximization in (15.15) can be performed more efficiently: approximately via alternating maximization (Emery-Montemerlo

⁶ Because Q^* is a function of φ_t and δ_t , (15.13) has a slightly different form than (15.15). The former technically does not correspond to a CBG, while the latter does.

⁷ There is a subtle but important difference between $Q^*(\bar{\theta}_t, a)$ and $Q^*(I, \varphi_t, \bar{\theta}_t, \delta_t)$: the latter specifies the optimal value given *any* past joint policy φ_t while the former only specifies optimal value given that π^* is actually being followed.

et al, 2004), or exactly via heuristic search (Kumar and Zilberstein, 2010b; Oliehoek et al, 2010). Second, it is possible to reduce the number of histories under concern via pruning (Emery-Montemerlo et al, 2004), approximate clustering (Emery-Montemerlo et al, 2005) or lossless clustering (Oliehoek et al, 2009).

15.4.4.3 Multi-Agent A*

Since FSPC using \hat{Q} can be seen as a single trace in a search tree, a natural idea is to allow for back-tracking and perform a full heuristic search as in *multi-agent A** (MAA*) (Szer et al, 2005), illustrated in Figure 15.7b.

MAA* performs an A* search over past joint policies φ_t . It computes a heuristic value $\hat{V}(\varphi_t)$ by taking $V^{0 \dots t-1}(\varphi_t)$, the actual expected reward over the first t stages, and adding $\hat{V}^{t \dots h-1}$, a heuristic value for the remaining $h-t$ stages. When the heuristic is *admissible*—a guaranteed overestimation—so is $\hat{V}(\varphi_t)$. MAA* performs standard A* search (Russell and Norvig, 2003): it maintains an open list P of partial joint policies φ_t and their heuristic values $\hat{V}(\varphi_t)$. On every iteration MAA* selects the highest ranked φ_t and expands it, generating and heuristically evaluating all $\varphi_{t+1} = \langle \varphi_t \circ \delta_t \rangle$ and placing them in P. When using an admissible heuristic, the heuristic values $\hat{V}(\varphi_{t+1})$ of the newly expanded policies are an upper bound to the true values and any lower bound \underline{y}^* that has been found can be used to prune P. The search ends when the list becomes empty, at which point an optimal fully specified joint policy has been found.

There is a direct relation between MAA* and the optimal value functions described in the previous section: V^* given by (15.14) is the optimal heuristic $\hat{V}^{t \dots h-1}$ (note that V^* only specifies reward from stage t onward).

MAA* suffers from the same problem as FSPC via CBGs: the number of δ_t grows doubly exponential with t , which means that the number of children of a node grows doubly exponential in its depth. In order to mitigate the problem, it is possible to apply lossless clustering (Oliehoek et al, 2009), or to try and avoid the expansion of all child nodes by incrementally expanding nodes only when needed (Spaan et al, 2011).

15.4.4.4 Generalized MAA*

Even though Figure 15.7 shows a clear relation between FSPC and MAA*, it is not directly obvious how they relate: the former solves CBGs, while the latter performs heuristic search. Generalized MAA* (GMAA*) (Oliehoek et al, 2008b) unifies these two approaches by making explicit the ‘Expand’ operator.

Algorithm 26 shows GMAA*. When the Select operator selects the highest ranked φ_t and when the Expand operator works as described for MAA*, GMAA* simply is MAA*. Alternatively, the Expand operator can construct a CBG $B(I, \varphi_t)$ for which all joint CBG-policies δ_t are evaluated. These can then be used to

```

Initialize:  $\underline{v}^* \leftarrow -\infty$ ,  $P \leftarrow \{\varphi_0 = ()\}$ 
repeat
   $\varphi_t \leftarrow \text{Select}(P)$ 
   $\Phi_{\text{Expand}} \leftarrow \text{Expand}(I, \varphi_t)$ 
  if  $\Phi_{\text{Expand}}$  contains full policies  $\Pi_{\text{Expand}} \subseteq \Phi_{\text{Expand}}$  then
     $\pi' \leftarrow \arg \max_{\pi \in \Pi_{\text{Expand}}} V(\pi)$ 
    if  $V(\pi') > \underline{v}^*$  then
       $\underline{v}^* \leftarrow V(\pi')$  {found new lower bound}
       $\pi^* \leftarrow \pi'$ 
       $P \leftarrow \{\varphi \in P \mid \widehat{V}(\varphi) > \underline{v}^*\}$  {prune P}
       $\Phi_{\text{Expand}} \leftarrow \Phi_{\text{Expand}} \setminus \Pi_{\text{Expand}}$  {remove full policies}
     $P \leftarrow (P \setminus \varphi_t) \cup \{\varphi \in \Phi_{\text{Expand}} \mid \widehat{V}(\varphi) > \underline{v}^*\}$  {remove processed/add new  $\varphi$ }
  until P is empty

```

Algorithm 26. (Generalized) MAA^{*}

construct a new set of partial policies $\Phi_{\text{Expand}} = \{\langle \varphi_t \circ \delta_t \rangle\}$ and their heuristic values. This corresponds to MAA^{*} reformulated to work on CBGs. It can be shown that when using a particular form of \widehat{Q} (including the mentioned heuristics Q_{MDP} , Q_{POMDP} and Q_{BG}), the approaches are identical (Oliehoek et al, 2008b). GMAA^{*} can also use an Expand operator that does not construct all new partial policies, but only the best-ranked one, $\Phi_{\text{Expand}} = \{\langle \varphi_t \circ \delta_t^* \rangle\}$. As a result the open list P will never contain more than one partial policy and behavior reduces to FSPC. A generalization called k -GMAA^{*} constructs the k best-ranked partial policies, allowing to trade off computation time and solution quality. Clustering of histories can also be applied in GMAA^{*}, but only lossless clustering will preserve optimality.

15.4.5 Backwards Approach: Dynamic Programming

The forward approach to Dec-POMDPs incrementally builds policies from the first stage $t = 0$ to the last $t = h - 1$. Prior to doing this, a Q-value function (optimal Q^* or approximate \widehat{Q}) needs to be computed. This computation itself, the dynamic program represented in Theorem 15.2, starts with the last stage and works its way back. The resulting optimal values correspond to the expected values of a joint decision rule and continuing optimally afterwards. That is, in the light of (15.10) this can be interpreted as the computation of the value for a subset of optimal (useful) joint sub-tree policies.

This section treats dynamic programming (DP) for Dec-POMDPs (Hansen, Bernstein, and Zilberstein, 2004). This method also works backwards, but rather than computing a Q-value function, it directly computes a set of useful sub-tree policies.

15.4.5.1 Dynamic Programming for Dec-POMDPs

The core idea of DP is to incrementally construct sets of longer sub-tree policies for the agents: starting with a set of one-step-to-go ($\tau = 1$) sub-tree policies (actions) that can be executed at the last stage, construct a set of 2-step policies to be executed at $h - 2$, etc. That is, DP constructs $Q_{\tau=1}^i, Q_{\tau=2}^i, \dots, Q_{\tau=h}^i$ for all agents i . When the last backup step is completed, the optimal policy can be found by evaluating all induced joint policies $\pi \in Q_{\tau=h}^1 \times \dots \times Q_{\tau=h}^n$ for the initial belief I as described in Section 15.3.4.

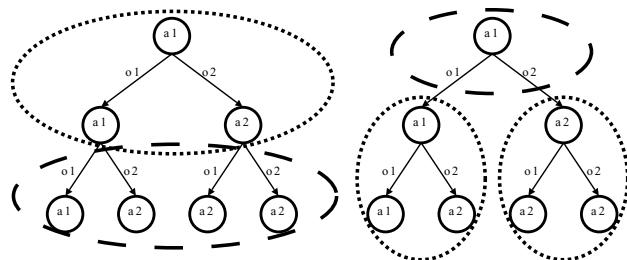


Fig. 15.8 Difference between policy construction in MAA* (left) and dynamic programming (right) for an agent with actions a, \bar{a} and observations o, \bar{o} . Dashed components are newly generated, dotted components result from the previous iteration.

DP formalizes this idea using *backup* operations that construct $Q_{\tau=k+1}^i$ from $Q_{\tau=k}^i$. For instance, the right side of Figure 15.8 shows how $q_{\tau=3}^i$, a 3-steps-to-go sub-tree policy, is constructed from two $q_{\tau=2}^i \in Q_{\tau=2}^i$. In general, a one step extended policy $q_{\tau=k+1}^i$ is created by selecting a sub-tree policy for each observation and an action for the root. An exhaustive backup generates *all* possible $q_{\tau=k+1}^i$ that have policies from the previously generated set $Q_{\tau=k}^i \in Q_{\tau=k}^i$ as their sub-trees. We will denote the sets of sub-tree policies resulting from exhaustive backup for each agent i by $Q_{\tau=k+1}^{e,i}$.

Unfortunately, sets of sub-tree policies maintained grow doubly exponentially with k .⁸ To counter this source of intractability, it is possible to prune dominated sub-tree policies from $Q_{\tau=k}^{e,i}$, resulting in smaller maintained sets $Q_{\tau=k}^{m,i}$ (Hansen et al, 2004). The value of a $q_{\tau=k}^i$ depends on the probability distribution over states when it is started (at stage $t = h - k$) as well as the probability with which the other agents $j \neq i$ select their sub-tree policies. Therefore, a $q_{\tau=k}^i$ is dominated if it is not maximizing at any point in the *multi-agent belief space*: the simplex over $S \times Q_{\tau=k}^{m,-i}$. It is possible to test for dominance by linear programming. Removal of a dominated sub-tree policy $q_{\tau=k}^i$ of an agent i may cause a $q_{\tau=k}^j$ of another agent j to become dominated. Therefore DP iterates over agents until no further pruning is possible,

⁸ Since the $q_{\tau=k}^i$ are essentially full policies for the horizon- k problem, their number is doubly exponentially in k .

a procedure known as *iterated elimination of dominated policies* (Osborne and Rubinstein, 1994).

In practice, the pruning step in DP often is not able to sufficiently reduce the maintained sets to make the approach tractable for larger problems. *Bounded DP (BDP)* can be used to compute a bounded approximation (Amato et al, 2007). It performs more aggressive ϵ -pruning: a $q_{\tau=k}^i$ that is maximizing in some region of the multi-agent belief space, but improves the value in this region by at most ϵ , is also pruned. Because iterated elimination using ϵ -pruning can still lead to an unbounded reduction in value, BDP performs one iteration of ϵ -pruning, followed by iterated elimination using normal pruning.

Even when many sub-tree policies can be pruned, DP can run into problems during the exhaustive backup. *Incremental policy generation (IPG)* is a technique to mitigate this problem by performing a one-step state reachability analysis (Amato et al, 2009). During the back up of sub-trees for an agent i , IPG analyzes the set of states $S_{\langle a^i, o^i \rangle}$ that have non-zero probability after each $\langle a^i, o^i \rangle$ -pair (a particular observation may exclude many states). Subsequently, in constructing the set $Q_{\tau=k+1}^{e,i}$, only sub-tree policies that are non-dominated for $S_{\langle a^i, o^i \rangle}$ are selected for action a^i and observation o^i . This can lead to much smaller sets of sub-tree policies.

An additional difficulty in DP is that, in order to perform pruning, all the $V(s, q_{\tau=k})$ values need to be computed and stored, which takes $|Q_{\tau=k}^{e,\dagger}|^n \times |S|$ real numbers. As such, DP runs out of memory well before it runs out of time. In order to address this problem Bouali and Chaib-draa (2008) represent these values more compactly by making use of a *sequence form* (Koller et al, 1994) representation. A disadvantage is that this approach can lead to keeping dominated policies, however. As such there is a trade-off between space required to store the values for all sub-tree policies and the number of sub-tree policies.

15.4.5.2 Point-Based DP

DP only removes $q_{\tau=k}^i$ that are not maximizing at *any point* in the multi-agent belief space. *Point-based DP (PBDP)* (Szer and Charillet, 2006) proposes to improve pruning of the set $Q_{\tau=k}^{e,i}$ by considering only a subset $\mathcal{B}^i \subset \mathcal{P}(S \times Q_{\tau=k}^{-i})$ of *reachable* multi-agent belief points. Only those $q_{\tau=k}^i$ that maximize the value at some $b^i \in \mathcal{B}^i$ are kept. The definition of reachable is slightly involved.

Definition 15.9. A multi-agent belief point b_t^i is *reachable* if there exists a probability distribution $P(s_t, \bar{\theta}_t^{-i} | I, \varphi_t)$ (for any deterministic φ_t) and an induced mapping $\Gamma_t^{-i} = \langle \Gamma_t^j \rangle_{j \neq i}$ with $\Gamma_t^j : \bar{\Theta}_t^j \rightarrow Q_{\tau=k}^j$ that result in b_t^i .

That is, a belief point b^i is reachable if there is a past joint policy that will result in the appropriate distribution over states and AOHs of other agents such that, when combined with a mapping of those AOHs to sub-tree policies, b^i is the resulting distribution over states and sub-tree policies.

PBDP can be understood in the light of (15.10). Suppose that the range of the Γ_t^i are restricted to the sets generated by exhaustive backup: $\Gamma_t^i : \bar{\Theta}_t^i \rightarrow Q_{\tau=k}^{e,i}$. Solving (15.10) for a past joint policy φ_t will result in Γ_t^* which will specify, for all agents, all the useful sub-tree policies $q_{\tau=k}^i \in Q_{\tau=k}^{e,i}$ given φ_t . Solving (15.10) for all φ_t will result in the set of all potentially useful $q_{\tau=k}^i \in Q_{\tau=k}^{e,i}$.

Given a φ_t and a Γ_t^{-i} , (15.10) can be rewritten as a maximization from the perspective of agent i to compute its best response:⁹

$$BR^i(\bar{\theta}_t^i, \varphi_t, \Gamma_t^{-i}) = \arg \max_{q_{\tau=k}^i} \sum_{\bar{\theta}_t^{-i}} P(s_t, \bar{\theta}_t^{-i} | \bar{\theta}_t^i, I, \varphi_t) V(s_t, \langle \Gamma_t^{-i}(\bar{\theta}_t^{-i}), q_{\tau=k}^i \rangle). \quad (15.16)$$

That is, given φ_t and Γ_t^{-i} , each $\bar{\theta}_t^i$ generates a multi-agent belief point, for which (15.16) performs the maximization. The set $Q_{\tau=k}^{m,i} := \{BR^i(\bar{\theta}_t^i, \varphi_t, \Gamma_t^{-i})\}$ of best responses for all φ_t , Γ_t^{-i} and $\bar{\theta}_t^i$, contains all non-dominated sub-tree policies, thus yielding an exact pruning step.

PBDP uses the initial belief to overcome the need to test for dominance over the entire multi-agent belief space. It can also result in more pruning, since it avoids maintaining sub-tree policies that are maximizing in a part of this space that cannot be reached. Still, the operation described above is intractable because the number of $(\bar{\theta}_t^i, \varphi_t, \Gamma_t^{-i})$ is doubly exponential in t and because the maintained sets $Q_{\tau=k}^{m,i}$ can still grow doubly exponentially.

15.4.5.3 Memory-Bounded DP

Memory-bounded DP (MBDP) (Seuken and Zilberstein, 2007b) is an approximate method that addresses these complexity issues by making two approximations. This first approximation is the assumption that the joint sub-tree policies that are maximizing for a joint belief are likely to specify good candidate individual sub-tree policies. I.e., instead of performing (15.16) to compute candidate sub-tree policies MBDP performs:

$$\forall_{\bar{\theta}_t} \quad q_{\tau=k}^{\bar{\theta}_t} = \arg \max_{q_{\tau=k} \in Q_{\tau=k}^e} V(I, \bar{\theta}_t, q_{\tau=k}), \quad (15.17)$$

where $Q_{\tau=k}^e \triangleq Q_{\tau=k}^{e,1} \times \dots \times Q_{\tau=k}^{e,n}$ is the set of $q_{\tau=k}$ induced by the sets exhaustively backed-up sub-trees $Q_{\tau=k}^{e,i}$. If a $q_{\tau=k}^i$ is not part of any $q_{\tau=k}^{\bar{\theta}_t}$, it is assumed to be dominated. Note that $V(I, \bar{\theta}_t, q_{\tau=k})$, defined by (15.7), only depends on $\bar{\theta}_t$ through the joint beliefs b it induces, so (15.17) only has to be evaluated for distinct b . Also note that this maximization is surprising: it was explained in Section 15.4.3.1 that performing this maximization for a particular node of the AOH tree is not possible. The difference here is that MBDP will not use the found $q_{\tau=k}^{\bar{\theta}_t}$ as the joint sub-tree policy for $\bar{\theta}_t$ (which might result in a centralized joint policy), but rather uses it to construct sets of *individual* candidate $q_{\tau=k}^i$.

⁹ The summation over states comes from substituting (15.7) for $V(I, \bar{\theta}_t, q_{\tau=k})$.

The second approximation is that MBDP maintains sets $\mathcal{Q}_{\tau=k}^{m,i}$ of a fixed size, M , which has two main consequences. First, the size of the candidate sets $\mathcal{Q}_{\tau=k}^{e,i}$ formed by exhaustive backup is $O(|A^\dagger| |M|^{|O^\dagger|})$, which clearly does not depend on the horizon. Second, (15.17) does not have to be evaluated for *all* distinct b ; rather MBDP *samples* M joint belief points b on which (15.17) is evaluated.¹⁰ To perform this sampling, MBDP uses heuristic policies.

In order to perform the maximization in (15.17), MBDP loops over the $|\mathcal{Q}_{\tau=k}^e| = O(|A^\dagger|^n |M^n|^{|O^\dagger|})$ joint sub-tree policies for each of the sampled belief points. To reduce the burden of this complexity, many papers have proposed new methods for performing this point-based backup operation (Seuken and Zilberstein, 2007a; Carlin and Zilberstein, 2008; Dibangoye et al, 2009; Amato et al, 2009; Wu et al, 2010a). This backup corresponds to solving a CBG for each joint action (Kumar and Zilberstein, 2010b; Oliehoek et al, 2010).

Finally, sample-based extensions have been proposed (Wu et al, 2010c,b). These use sampling to evaluate the quantities $V(s, q_{\tau=k})$ and use particle representations for the sampled joint beliefs.

15.4.6 Other Finite-Horizon Methods

There are a few other approaches for finite-horizon Dec-POMDPs, which we will only briefly describe here. Aras et al (2007) proposed a mixed integer linear programming formulation for the optimal solution of finite-horizon Dec-POMDPs, based on representing the set of possible policies for each agent in *sequence form* (Koller and Pfeffer, 1997). In this representation, a policy for an agent i is represented as a subset of the set of *sequences* (roughly corresponding to action-observation histories) for the agent. As such the problem can be interpreted as a combinatorial optimization problem and solved with a mixed integer linear program.

The fact that solving a Dec-POMDP can be approached as a combinatorial optimization problem was also recognized by approaches based on cross-entropy optimization (Oliehoek et al, 2008a) and genetic algorithms (Eker and Akin, 2008).

15.5 Further Topics

This section provides pointers to some further topics in Dec-POMDPs.

15.5.1 Generalization and Special Cases

The generalization of the Dec-POMDP is the *partially observable stochastic game (POSG)*. It has the same components as a Dec-POMDP, except that it specifies a

¹⁰ If evaluation of (15.17) leads to duplicate $q_{\tau=k}^i$ more samples may be necessary.

collection of reward functions: one for each agent. A POSG assumes self-interested agents that maximize their individual expected cumulative reward. The consequence of this is that there is no longer a simple concept of optimal joint policy. Rather the joint policy should be a Nash Equilibrium (NE), and preferably a Pareto optimal NE. However, there is no clear way to identify the best one. Moreover, such an NE is only guaranteed to exist in randomized policies (for a finite POSG), which means that it is no longer possible to perform brute-force policy evaluation. Also, search methods based on alternating maximization are no longer guaranteed to converge for POSGs. The (not point-based) dynamic programming method, discussed in Section 15.4.5.1, applies to POSGs since it finds the set of non-dominated policies for each agent.

Because of the negative complexity results for Dec-POMDPs, much research has focused on special cases to which pointers are given below. For a more comprehensive overview the reader is referred to the texts by Pynadath and Tambe (2002); Goldman and Zilberstein (2004); Seuken and Zilberstein (2008).

Some of the special cases are formed by different degrees of observability. These range from *fully-* or *individually observable* as in a multi-agent MDP (Boutilier, 1996) to *non-observable*. In the non-observable case agents use open-loop policies and solving it is easier from a complexity point of view (NP-complete, Pynadath and Tambe 2002). Between these two extremes there are partially observable problems. One more special case has been identified, namely the *jointly observable* case, where not the individual, but the joint observation identifies the true state. A jointly observable Dec-POMDP is referred to as a *Dec-MDP*, which is a non-trivial subclass of the Dec-POMDP for which the NEXP-completeness result holds (Bernstein et al, 2002).

Other research has tried to exploit structure in states, transitions and reward. For instance, many approaches are based on special cases of *factored Dec-POMDPs*. A factored Dec-POMDP (Oliehoek et al, 2008c) is a Dec-POMDP in which the state space is factored, i.e., a state $s = \langle x_1, \dots, x_k \rangle$ is specified as an assignment to a number of state variables or *factors*. For factored Dec-POMDPs the transition and reward models can often be specified much more compactly by making use of Bayesian networks and additive reward decomposition (the total reward is the sum of a number of ‘smaller’ reward functions, specified over a subset of agents). Many special cases have tried to exploit independence between agents by partitioning the set of state factors into individual states s_i for each agent.

One such example is the *transition- and observation-independent (TOI) Dec-MDP* (Becker et al, 2004b; Wu and Durfee, 2006) that assumes each agent i has its own MDP with local states s_i and transitions, but that these MDPs are coupled through certain *events* in the reward function: some combinations of joint actions and joint states will cause extra reward (or penalty). This work introduced the idea that in order to compute a best response against a policy π^j , an agent i may not need to reason about all the details of π^j , but can use a more abstract representation of the influence of π^j on itself. This core idea was also used in *event-driven (ED) Dec-MDPs* (Becker et al, 2004a) that model settings in which the rewards are independent, but there are certain events that cause transition dependencies. Mostafa and Lesser (2009) introduced the EDI-CR, a type of Dec-POMDP that generalizes TOI

and ED-Dec-MDPs. Recently the idea of abstraction has been further explored by Witwicki and Durfee (2010), resulting in a more general formulation of influence-based policy abstraction for a more general sub-class of the factored Dec-POMDP called *temporally decoupled Dec-POMDP (TD-POMDP)* that also generalizes TOI- and ED-Dec-MDPs (Witwicki, 2011). While much more general than TOI Dec-MDPs (e.g., the local states of agents can overlap) the TD-POMDP is still restrictive as it does not allow multiple agents to have direct influence on the same state factor.

Finally, there has been a body of work on *networked distributed POMDPs (ND-POMDPs)* (Nair et al, 2005; Kim et al, 2006; Varakantham et al, 2007; Marecki et al, 2008; Kumar and Zilberstein, 2009; Varakantham et al, 2009). ND-POMDPs can be understood as factored Dec-POMDPs with TOI and additively factored reward functions. For this model, it was shown that the value function $V(\pi)$ can be additively factored as well. As a consequence, it is possible to apply many ideas from distributed constraint optimization in order to optimize the value more efficiently. As such ND-POMDPs have been shown to scale to moderate numbers (up to 20) of agents. These results were extended to general factored Dec-POMDPs by Oliehoek et al (2008c). In that case, the amount of independence depends on the stage of the process; earlier stages are typically fully coupled limiting exact solutions to small horizons and few (three) agents. Approximate solutions, however, were shown to scale to hundreds of agents (Oliehoek, 2010).

15.5.2 Infinite-Horizon Dec-POMDPs

The main focus of this chapter has been on finding solution methods for finite-horizon Dec-POMDPs. There also has been quite a bit of work on infinite-horizon Dec-POMDPs, some of which is summarized here.

The infinite-horizon case is substantially different from the finite-horizon case. For instance, the infinite-horizon problem is undecidable (Bernstein et al, 2002), which is a direct result of the undecidability of (single-agent) POMDPs over an infinite horizon (Madani et al, 1999). This can be understood by thinking about the representations of policies; in the infinite-horizon case the policy trees themselves should be infinite and clearly there is no way to represent that in a finite amount of memory.

As a result, research on infinite-horizon Dec-POMDPs has focused on approximate methods that use finite policy representations. A common choice is to use finite state controllers (FSCs). A side-effect of limiting the amount of memory for the policy is that in many cases it can be beneficial to allow stochastic policies (Singh et al, 1994). Most research in this line of work has proposed methods that incrementally improve the quality of the controller. For instance, Bernstein et al (2009) propose a policy iteration algorithm that computes an ϵ -optimal solution by iteratively performing backup operations on the FSCs. These backups, however, grow the size of the controller exponentially. While value-preserving transformations may reduce the size of the controller, the controllers can still grow unboundedly.

One idea to overcome this problem is *bounded policy iteration (BPI)* for Dec-POMDPs (Bernstein et al, 2005). BPI keeps the number of nodes of the FSCs fixed by applying bounded backups. BPI converges to a local optimum *given a particular controller size*. Amato et al (2010) also consider finding an optimal joint policy given a particular controller size, but instead propose a non-linear programming (NLP) formulation. While this formulation characterizes the true optimum, solving the NLP exactly is intractable. However, approximate NLP solvers have shown good results in practice.

Finally, a recent development has been to address infinite-horizon Dec-POMDPs via the planning-as-inference paradigm (Kumar and Zilberstein, 2010a). Pajarinen and Peltonen (2011) extended this approach to factored Dec-POMDPs.

15.5.3 Reinforcement Learning

A next related issue is the more general setting of multi-agent reinforcement learning (MARL). That is, this chapter has focused on the task of planning given a model. In a MARL setting however, the agents do not have access to such a model. Rather, the model will have to be learned on-line (model-based MARL) or the agents will have to use model-free methods. While there is a great deal of work on MARL in general (Buşoniu et al, 2008), MARL in Dec-POMDP-like settings has received little attention.

Probably one of the main reasons for this gap in literature is that it is hard to properly define the setup of the RL problem in these partially observable environments with multiple agents. For instance, it is not clear when or how the agents will observe rewards.¹¹ Moreover, even when the agents can observe the state, convergence of MARL is not well-understood: from the perspective of one agent, the environment has become non-stationary since the other agent is also learning, which means that convergence guarantees for single-agent RL no longer hold. Claus and Boutilier (1998) argue that, in a cooperative setting, independent Q-learners are guaranteed to converge to a local optimum, but not the optimal solution. Nevertheless, this method has on occasion been reported to be successful in practice (e.g., Crites and Barto, 1998) and theoretical understanding of convergence of individual learners is progressing (e.g., Tuyls et al, 2006; Kaisers and Tuyls, 2010; Wunder et al, 2010). There are coupled learning methods (e.g., Q-learning using the joint action space) that will converge to an optimal solution (Vlassis, 2007). However, all forms of coupled learning are precluded in the true Dec-POMDP setting: such algorithms require either full observation of the state and actions of other agents, or communication of all the state information.

Concluding this section we will provide pointers to a few notable approaches to RL in Dec-POMDP-like settings. Peshkin et al (2000) introduced *decentralized*

¹¹ Even in a POMDP, the agent is not assumed to have access to the immediate rewards, since they can convey hidden information about the states.

gradient ascent policy search (DGAPS), a method for MARL in partially observable settings based on gradient descent. DGAPS represents individual policies using FSCs and assumes that agents observe the global rewards. Based in this, it is possible for each agent to independently update its policy in the direction of the gradient with respect to the return, resulting in a locally optimal joint policy. This approach was extended to learn policies for self-configurable modular robots (Varshavskaya et al, 2008). Chang et al (2004) also consider decentralized RL assuming that the global rewards are available to the agents. In their approach, these global rewards are interpreted as individual rewards, corrupted by noise due to the influence of other agents. Each agent explicitly tries to estimate the individual reward using Kalman filtering and performs independent Q-learning using the filtered individual rewards. The method by Wu et al (2010b) is closely related to RL since it does not need a model as input. It does, however, needs access to a simulator which can be initialized to specific states. Moreover, the algorithm itself is centralized, as such it is not directly suitable for on-line RL.

Finally, there are MARL methods for partially observed decentralized settings that require only limited amounts of communication. For instance, Boyan and Littman (1993) considered decentralized RL for a packet routing problem. Their approach, Q-routing, performs a type of Q-learning where there is only limited local communication: neighboring nodes communicate the expected future waiting time for a packet. Q-routing was extended to mobile wireless networks by Chang and Ho (2004). A similar problem, distributed task allocation, is considered by Abdallah and Lesser (2007). In this problem there also is a network, but now agents do not send communication packets, but rather tasks to neighbors. Again, communication is only local. Finally, in some RL methods for multi-agent MDPs (i.e., coupled methods) it is possible to have agents observe a subset of state factors if they have the ability to communicate *locally* (Guestrin et al, 2002; Kok and Vlassis, 2006).

15.5.4 Communication

The Dec-POMDP has been extended to explicitly incorporate communication actions, and observations. The resulting model, the Dec-POMDP-Com (Goldman and Zilberstein, 2003, 2004) includes a set of messages that can be sent by each agent and a cost function that specifies the cost of sending each message. The goal in a Dec-POMDP-Com is to:

“find a joint policy that maximizes the expected total reward over the finite horizon. Solving for this policy embeds the *optimal meaning* of the messages chosen to be communicated” — Goldman and Zilberstein (2003)

That is, in this perspective the semantics of the communication actions become part of the optimization problem (Xuan et al, 2001; Goldman and Zilberstein, 2003; Spaan et al, 2006; Goldman et al, 2007).

One can also consider the case where messages have fixed semantics. In such a case the agents need a mechanism to process these semantics. For instance, when the agents share their local observations, each agent maintains a joint belief and performs an update of this joint belief, rather than maintaining the list of observations. It was shown by Pynadath and Tambe (2002) that under cost-free communication, a joint communication policy that shares the local observations at each stage is optimal. Much research has investigated sharing local observations in models similar to the Dec-POMDP-Com (Ooi and Wornell, 1996; Pynadath and Tambe, 2002; Nair et al, 2004; Becker et al, 2005; Roth et al, 2005b,a; Spaan et al, 2006; Oliehoek et al, 2007; Roth et al, 2007; Goldman and Zilberstein, 2008; Wu et al, 2011).

A final note is that, although models with explicit communication seem more general than the models without, it is possible to transform the former to the latter. That is, a Dec-POMDP-Com can be transformed to a Dec-POMDP (Goldman and Zilberstein, 2004; Seuken and Zilberstein, 2008).

Acknowledgements. I would like to thank Leslie Kaelbling and Shimon Whiteson for the valuable feedback they provided and the reviewers for their insightful comments. Research supported by AFOSR MURI project #FA9550-09-1-0538.

References

- Abdallah, S., Lesser, V.: Multiagent reinforcement learning and self-organization in a network of agents. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 172–179 (2007)
- Amato, C., Carlin, A., Zilberstein, S.: Bounded dynamic programming for decentralized POMDPs. In: Proc. of the AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains, MSDM (2007)
- Amato, C., Dibangoye, J.S., Zilberstein, S.: Incremental policy generation for finite-horizon DEC-POMDPs. In: Proc. of the International Conference on Automated Planning and Scheduling, pp. 2–9 (2009)
- Amato, C., Bernstein, D.S., Zilberstein, S.: Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. Autonomous Agents and Multi-Agent Systems 21(3), 293–320 (2010)
- Aras, R., Dutech, A., Charpillet, F.: Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In: Proc. of the International Conference on Automated Planning and Scheduling (2007)
- Becker, R., Zilberstein, S., Lesser, V.: Decentralized Markov decision processes with event-driven interactions. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 302–309 (2004a)
- Becker, R., Zilberstein, S., Lesser, V., Goldman, C.V.: Solving transition independent decentralized Markov decision processes. Journal of Artificial Intelligence Research 22, 423–455 (2004b)
- Becker, R., Lesser, V., Zilberstein, S.: Analyzing myopic approaches for multi-agent communication. In: Proc. of the International Conference on Intelligent Agent Technology, pp. 550–557 (2005)

- Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27(4), 819–840 (2002)
- Bernstein, D.S., Hansen, E.A., Zilberstein, S.: Bounded policy iteration for decentralized POMDPs. In: Proc. of the International Joint Conference on Artificial Intelligence, pp. 1287–1292 (2005)
- Bernstein, D.S., Amato, C., Hansen, E.A., Zilberstein, S.: Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research* 34, 89–132 (2009)
- Boularias, A., Chaib-draa, B.: Exact dynamic programming for decentralized POMDPs with lossless policy compression. In: Proc. of the International Conference on Automated Planning and Scheduling (2008)
- Boutilier, C.: Planning, learning and coordination in multiagent decision processes. In: Proc. of the 6th Conference on Theoretical Aspects of Rationality and Knowledge, pp. 195–210 (1996)
- Boyan, J.A., Littman, M.L.: Packet routing in dynamically changing networks: A reinforcement learning approach. In: Advances in Neural Information Processing Systems, vol. 6, pp. 671–678 (1993)
- Bušoniu, L., Babuška, R., De Schutter, B.: A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 38(2), 156–172 (2008)
- Carlin, A., Zilberstein, S.: Value-based observation compression for DEC-POMDPs. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 501–508 (2008)
- Chang, Y.H., Ho, T.: Mobilized ad-hoc networks: A reinforcement learning approach. In: Proceedings of the First International Conference on Autonomic Computing, pp. 240–247 (2004)
- Chang, Y.H., Ho, T., Kaelbling, L.P.: All learning is local: Multi-agent learning in global reward games. In: Advances in Neural Information Processing Systems, vol. 16 (2004)
- Claus, C., Boutilier, C.: The dynamics of reinforcement learning in cooperative multiagent systems. In: Proc. of the National Conference on Artificial Intelligence, pp. 746–752 (1998)
- Cogill, R., Rotkowitz, M., Roy, B.V., Lall, S.: An approximate dynamic programming approach to decentralized control of stochastic systems. In: Proc. of the 2004 Allerton Conference on Communication, Control, and Computing (2004)
- Crites, R.H., Barto, A.G.: Elevator group control using multiple reinforcement learning agents. *Machine Learning* 33(2-3), 235–262 (1998)
- Dibangoye, J.S., Mouaddib, A.I., Chai-draa, B.: Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 569–576 (2009)
- Eker, B., Akin, H.L.: Using evolution strategies to solve DEC-POMDP problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications* (2008)
- Emery-Montemerlo, R., Gordon, G., Schneider, J., Thrun, S.: Approximate solutions for partially observable stochastic games with common payoffs. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 136–143 (2004)
- Emery-Montemerlo, R., Gordon, G., Schneider, J., Thrun, S.: Game theoretic control for robot teams. In: Proc. of the IEEE International Conference on Robotics and Automation, pp. 1175–1181 (2005)

- Goldman, C.V., Zilberstein, S.: Optimizing information exchange in cooperative multi-agent systems. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 137–144 (2003)
- Goldman, C.V., Zilberstein, S.: Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research* 22, 143–174 (2004)
- Goldman, C.V., Zilberstein, S.: Communication-based decomposition mechanisms for decentralized MDPs. *Journal of Artificial Intelligence Research* 32, 169–202 (2008)
- Goldman, C.V., Allen, M., Zilberstein, S.: Learning to communicate in a decentralized environment. *Autonomous Agents and Multi-Agent Systems* 15(1), 47–90 (2007)
- Guestrin, C., Lagoudakis, M., Parr, R.: Coordinated reinforcement learning. In: Proc. of the International Conference on Machine Learning, pp. 227–234 (2002)
- Hansen, E.A., Bernstein, D.S., Zilberstein, S.: Dynamic programming for partially observable stochastic games. In: Proc. of the National Conference on Artificial Intelligence, pp. 709–715 (2004)
- Kaisers, M., Tuyls, K.: Frequency adjusted multi-agent Q-learning. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 309–316 (2010)
- Kim, Y., Nair, R., Varakantham, P., Tambe, M., Yokoo, M.: Exploiting locality of interaction in networked distributed POMDPs. In: Proc. of the AAAI Spring Symposium on Distributed Plan and Schedule Management (2006)
- Kok, J.R., Vlassis, N.: Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research* 7, 1789–1828 (2006)
- Koller, D., Pfeffer, A.: Representations and solutions for game-theoretic problems. *Artificial Intelligence* 94(1-2), 167–215 (1997)
- Koller, D., Megiddo, N., von Stengel, B.: Fast algorithms for finding randomized strategies in game trees. In: Proc. of the 26th ACM Symposium on Theory of Computing, pp. 750–759 (1994)
- Kumar, A., Zilberstein, S.: Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 561–568 (2009)
- Kumar, A., Zilberstein, S.: Anytime planning for decentralized POMDPs using expectation maximization. In: Proc. of Uncertainty in Artificial Intelligence (2010a)
- Kumar, A., Zilberstein, S.: Point-based backup for decentralized POMDPs: Complexity and new algorithms. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 1315–1322 (2010b)
- Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In: Proc. of the National Conference on Artificial Intelligence, pp. 541–548 (1999)
- Marecki, J., Gupta, T., Varakantham, P., Tambe, M., Yokoo, M.: Not all agents are equal: scaling up distributed POMDPs for agent networks. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 485–492 (2008)
- Mostafa, H., Lesser, V.: Offline planning for communication by exploiting structured interactions in decentralized MDPs. In: Proc. of 2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, pp. 193–200 (2009)
- Nair, R., Tambe, M., Marsella, S.: Role allocation and reallocation in multiagent teams: towards a practical analysis. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 552–559 (2003a)

- Nair, R., Tambe, M., Marsella, S.C.: Team Formation for Reformation in Multiagent Domains Like RoboCupRescue. In: Kaminka, G.A., Lima, P.U., Rojas, R. (eds.) RoboCup 2002: Robot Soccer World Cup VI, LNCS (LNAI), vol. 2752, pp. 150–161. Springer, Heidelberg (2003)
- Nair, R., Tambe, M., Yokoo, M., Pynadath, D.V., Marsella, S.: Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In: Proc. of the International Joint Conference on Artificial Intelligence, pp. 705–711 (2003c)
- Nair, R., Roth, M., Yohoo, M.: Communication for improving policy computation in distributed POMDPs. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 1098–1105 (2004)
- Nair, R., Varakantham, P., Tambe, M., Yokoo, M.: Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In: Proc. of the National Conference on Artificial Intelligence, pp. 133–139 (2005)
- Oliehoek, F.A.: Value-based planning for teams of agents in stochastic partially observable environments. PhD thesis, Informatics Institute, University of Amsterdam (2010)
- Oliehoek, F.A., Vlassis, N.: Q-value functions for decentralized POMDPs. In: Proc. of The International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 833–840 (2007)
- Oliehoek, F.A., Spaan, M.T.J., Vlassis, N.: Dec-POMDPs with delayed communication. In: AAMAS Workshop on Multi-agent Sequential Decision Making in Uncertain Domains (2007)
- Oliehoek, F.A., Kooi, J.F., Vlassis, N.: The cross-entropy method for policy search in decentralized POMDPs. *Informatica* 32, 341–357 (2008a)
- Oliehoek, F.A., Spaan, M.T.J., Vlassis, N.: Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research* 32, 289–353 (2008b)
- Oliehoek, F.A., Spaan, M.T.J., Whiteson, S., Vlassis, N.: Exploiting locality of interaction in factored Dec-POMDPs. In: Proc. of The International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 517–524 (2008)
- Oliehoek, F.A., Whiteson, S., Spaan, M.T.J.: Lossless clustering of histories in decentralized POMDPs. In: Proc. of The International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 577–584 (2009)
- Oliehoek, F.A., Spaan, M.T.J., Dibangoye, J., Amato, C.: Heuristic search for identical payoff Bayesian games. In: Proc. of The International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 1115–1122 (2010)
- Ooi, J.M., Wornell, G.W.: Decentralized control of a multiple access broadcast channel: Performance bounds. In: Proc. of the 35th Conference on Decision and Control, pp. 293–298 (1996)
- Osborne, M.J., Rubinstein, A.: A Course in Game Theory. The MIT Press (1994)
- Pajarinen, J., Peltonen, J.: Efficient planning for factored infinite-horizon DEC-POMDPs. In: Proc. of the International Joint Conference on Artificial Intelligence (to appear, 2011)
- Paquet, S., Tobin, L., Chaib-draa, B.: An online POMDP algorithm for complex multiagent environments. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems (2005)
- Peshkin, L.: Reinforcement learning by policy search. PhD thesis, Brown University (2001)
- Peshkin, L., Kim, K.E., Meuleau, N., Kaelbling, L.P.: Learning to cooperate via policy search. In: Proc. of Uncertainty in Artificial Intelligence, pp. 307–314 (2000)
- Pynadath, D.V., Tambe, M.: The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research* 16, 389–423 (2002)

- Rabinovich, Z., Goldman, C.V., Rosenschein, J.S.: The complexity of multiagent systems: the price of silence. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 1102–1103 (2003)
- Roth, M., Simmons, R., Veloso, M.: Decentralized communication strategies for coordinated multi-agent policies. In: Parker, L.E., Schneider, F.E., Shultz, A.C. (eds.) *Multi-Robot Systems. From Swarms to Intelligent Automata*, vol. III, pp. 93–106. Springer, Heidelberg (2005a)
- Roth, M., Simmons, R., Veloso, M.: Reasoning about joint beliefs for execution-time communication decisions. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 786–793 (2005b)
- Roth, M., Simmons, R., Veloso, M.: Exploiting factored representations for decentralized execution in multi-agent teams. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 467–463 (2007)
- Russell, S., Norvig, P.: *Artificial Intelligence: A Modern Approach*, 2nd edn. Pearson Education (2003)
- Seuken, S., Zilberstein, S.: Improved memory-bounded dynamic programming for decentralized POMDPs. In: Proc. of Uncertainty in Artificial Intelligence (2007a)
- Seuken, S., Zilberstein, S.: Memory-bounded dynamic programming for DEC-POMDPs. In: Proc. of the International Joint Conference on Artificial Intelligence, pp. 2009–2015 (2007b)
- Seuken, S., Zilberstein, S.: Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems* 17(2), 190–250 (2008)
- Singh, S.P., Jaakkola, T., Jordan, M.I.: Learning without state-estimation in partially observable Markovian decision processes. In: Proc. of the International Conference on Machine Learning, pp. 284–292. Morgan Kaufmann (1994)
- Spaan, M.T.J., Gordon, G.J., Vlassis, N.: Decentralized planning under uncertainty for teams of communicating agents. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 249–256 (2006)
- Spaan, M.T.J., Oliehoek, F.A., Amato, C.: Scaling up optimal heuristic search in DEC-POMDPs via incremental expansion. In: Proc. of the International Joint Conference on Artificial Intelligence (to appear, 2011)
- Szer, D., Charpillet, F.: Point-based dynamic programming for DEC-POMDPs. In: Proc. of the National Conference on Artificial Intelligence (2006)
- Szer, D., Charpillet, F., Zilberstein, S.: MAA*: A heuristic search algorithm for solving decentralized POMDPs. In: Proc. of Uncertainty in Artificial Intelligence, pp. 576–583 (2005)
- Tuyls, K., Hoen, P.J., Vanschoenwinkel, B.: An evolutionary dynamical analysis of multiagent learning in iterated games. *Autonomous Agents and Multi-Agent Systems* 12(1), 115–153 (2006)
- Varakantham, P., Marecki, J., Yabu, Y., Tambe, M., Yokoo, M.: Letting loose a SPIDER on a network of POMDPs: Generating quality guaranteed policies. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems (2007)
- Varakantham, P., Young Kwak, J., Taylor, M.E., Marecki, J., Scerri, P., Tambe, M.: Exploiting coordination locales in distributed POMDPs via social model shaping. In: Proc. of the International Conference on Automated Planning and Scheduling (2009)
- Varshavskaya, P., Kaelbling, L.P., Rus, D.: Automated design of adaptive controllers for modular robots using reinforcement learning. *International Journal of Robotics Research* 27(3–4), 505–526 (2008)

- Vlassis, N.: A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool Publishers (2007)
- Witwicki, S.J.: Abstracting influences for efficient multiagent coordination under uncertainty. PhD thesis, University of Michigan, Ann Arbor, Michigan, USA (2011)
- Witwicki, S.J., Durfee, E.H.: Influence-based policy abstraction for weakly-coupled Dec-POMDPs. In: Proc. of the International Conference on Automated Planning and Scheduling, pp. 185–192 (2010)
- Wu, F., Zilberstein, S., Chen, X.: Point-based policy generation for decentralized POMDPs. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 1307–1314 (2010a)
- Wu, F., Zilberstein, S., Chen, X.: Rollout sampling policy iteration for decentralized POMDPs. In: Proc. of Uncertainty in Artificial Intelligence (2010b)
- Wu, F., Zilberstein, S., Chen, X.: Trial-based dynamic programming for multi-agent planning. In: Proc. of the National Conference on Artificial Intelligence, pp. 908–914 (2010c)
- Wu, F., Zilberstein, S., Chen, X.: Online planning for multi-agent systems with bounded communication. *Artificial Intelligence* 175(2), 487–511 (2011)
- Wu, J., Durfee, E.H.: Mixed-integer linear programming for transition-independent decentralized MDPs. In: Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems, pp. 1058–1060 (2006)
- Wunder, M., Littman, M.L., Babes, M.: Classes of multiagent Q-learning dynamics with epsilon-greedy exploration. In: Proc. of the International Conference on Machine Learning, pp. 1167–1174 (2010)
- Xuan, P., Lesser, V., Zilberstein, S.: Communication decisions in multi-agent cooperation: Model and experiments. In: Proc. of the International Conference on Autonomous Agents (2001)
- Zettlemoyer, L.S., Milch, B., Kaelbling, L.P.: Multi-agent filtering with infinitely nested beliefs. In: Advances in Neural Information Processing Systems, vol. 21 (2009)

Part V

Domains and Background

Chapter 16

Psychological and Neuroscientific Connections with Reinforcement Learning

Ashvin Shah

Abstract. The field of Reinforcement Learning (RL) was inspired in large part by research in animal behavior and psychology. Early research showed that animals can, through trial and error, learn to execute behavior that would eventually lead to some (presumably satisfactory) outcome, and decades of subsequent research was (and is still) aimed at discovering the mechanisms of this learning process. This chapter describes behavioral and theoretical research in animal learning that is directly related to fundamental concepts used in RL. It then describes neuroscientific research that suggests that animals and many RL algorithms use very similar learning mechanisms. Along the way, I highlight ways that research in computer science contributes to and can be inspired by research in psychology and neuroscience.

16.1 Introduction

In the late nineteenth century, the psychologist Edward L. Thorndike conducted experiments in which he placed a hungry animal (most famously a cat) in a “puzzle box,” the door of which could be opened only after a certain sequence of actions, such as pulling a chain and then pressing a lever, have been executed (Thorndike, 1911). Placed outside the box and in view of the animal was some food. A naive animal would exhibit many behaviors that had no effect on the door, such as batting at the door or even grooming itself. At some point, the animal might, by chance, pull the chain and later, also by chance, press the lever, after which the door would open and the animal could escape and consume the food. When that animal was again hungry and placed into the box, it would execute less of the useless behaviors and more of the ones that led to the opening door. After repeated trials, the animal could escape the box in mere seconds.

Ashvin Shah
Department of Psychology, University of Sheffield, Sheffield, UK
e-mail: ashvin@gmail.com

The animal's behavior is familiar to readers of this book as it describes well the behavior of a reinforcement learning (RL) agent engaged in a simple task. The basic problems the animal faces—and solves—in the puzzle box are those that an RL agent must solve: given no instruction and only a very coarse evaluation signal, how does an agent learn what to do and when to do it in order to better its circumstances? While RL is not intended to be a model of animal learning, animal behavior and psychology form a major thread of research that led to its development (Chapter 1, Sutton and Barto 1998). RL was also strongly influenced by the work of Harry Klopf (Klopf, 1982), who put forth the idea that hedonistic ("pleasure seeking") behavior emerges from hedonistic learning processes, including processes that govern the behavior of single neurons.

In this chapter I describe some of the early experimental work in animal behavior that started the field and developed the basic paradigms that are used even today, and psychological theories that were developed to explain observed behavior. I then describe neuroscience research aimed at discovering the brain mechanisms responsible for such behavior. Rather than attempt to provide an exhaustive review of animal learning and behavior and their underlying neural mechanisms in a single chapter, I focus on studies that are directly-related to fundamental concepts used in RL and that illustrate some of the experimental methodology. I hope that this focus will make clear the similarities—in some cases striking—between mechanisms used by RL agents and mechanisms thought to dictate many types of animal behavior. The fact that animals can solve problems we strive to develop artificial systems to solve suggests that a greater understanding of psychology and neuroscience can inspire research in RL and machine learning in general.

16.2 Classical (or Pavlovian) Conditioning

Prediction plays an important role in learning and control. Perhaps the most direct way to study prediction in animals is with classical conditioning, pioneered by Ivan Pavlov in Russia in the early 1900s. While investigating digestive functions of dogs, Pavlov noticed that some dogs that he had worked with before would salivate before any food was brought out. In what began as an attempt to account for this surprising behavior, Pavlov developed his theory of conditioned reflexes (Pavlov, 1927): mental processes (e.g., perception of an auditory tone) can cause a physiological reaction (salivation) that was previously thought to be caused only by physical processes (e.g., smell or presence of food in the mouth). Most famously, the sound of a ringing bell that reliably preceded the delivery of food eventually by itself caused the dog to salivate. This behavior can be thought of as an indication that the dog has learned to predict that food delivery will follow the ringing bell.

16.2.1 Behavior

While Pavlov's drooling dog is the enduring image of classical conditioning, a more studied system is the nictitating membrane (NM) of the rabbit eye (Gormezano et al, 1962), which is a thin "third eyelid" that closes to protect the eye. Typically, the rabbit is restrained and an air puff or a mild electric shock applied to the eye (the unconditioned stimulus, US) causes the NM to close (the unconditioned response, UR). If a neutral stimulus that does not by itself cause the NM to close (conditioned stimulus, CS), such as a tone or a light, is reliably presented to the rabbit before the US, eventually the CS itself causes the NM to close (the conditioned response, CR). Because the CR is acquired with repeated pairings of the CS and the US (the *acquisition* phase of an experiment), the US is often referred to as a *reinforcer*. The strength of the CR, often measured by how quickly the NM closes or the likelihood that it closes before the US, is a measure of the predictive strength of the CS for the animal. After the CR is acquired, if the CS is presented but the US is omitted (*extinction* phase), the strength of the CR gradually decreases.

Manipulations to the experimental protocol can give us a better idea of how such predictions are learned. Particularly instructive are manipulations in the timing of the CS relative to the US and how the use of multiple CSs affects the predictive qualities of each CS. (These manipulations are focused on in Sutton and Barto (1987) and Sutton and Barto (1981), which describe temporal difference models of classical conditioning.)

Two measures of timing between the CS and the subsequent US are (Figure 16.1, top): 1) interstimulus interval (ISI), which is the time between the onset of the CS

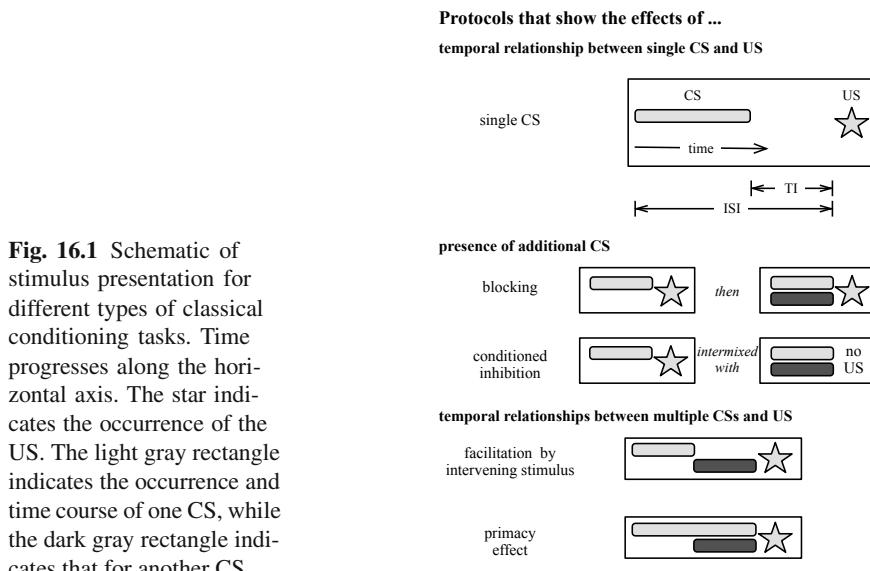


Fig. 16.1 Schematic of stimulus presentation for different types of classical conditioning tasks. Time progresses along the horizontal axis. The star indicates the occurrence of the US. The light gray rectangle indicates the occurrence and time course of one CS, while the dark gray rectangle indicates that for another CS.

and the onset of the US, and 2) trace interval (TI), which is the time between the offset of the CS and the onset of the US. A simple protocol uses a short and constant ISI and a zero-length TI (*delay conditioning*). For example, the tone is presented briefly (500 ms) and the air puff is presented at the end of the tone. When the TI is greater than zero (*trace conditioning*), acquisition and retention of the CR are hindered. If the ISI is zero, i.e., if the CS and US are presented at the same time, the CS is useless for prediction and the animal will not acquire a CR. There is an optimal ISI (about 250 ms for the NM response) after which the strength of the CR decreases gradually. The rate of decrease is greater in trace conditioning than it is in delay conditioning. In addition, the rate of acquisition decreases with an increase in ISI, suggesting that it is harder to predict temporally distant events.

The use of several CSs (*compound stimuli*) reveals that learning also depends on the animal's ability to predict the upcoming US. Figure 16.1, middle, illustrates example protocols in which one CS (e.g., a tone, referred to as CSA) is colored in light gray and the other (a light, CSB) is colored in dark gray. In *blocking* (Kamin, 1969), the US is paired with CSA alone and the animal acquires a CR. Afterwards, the simultaneous presentation of CSA and CSB is paired with the US for a block of trials. Subsequent presentation of CSB alone elicits no CR. Because CSA was already a predictor of the US, CSB holds no predictive power. In *conditioned inhibition*, two types of stimulus presentations are intermixed during training: CSA alone paired with the US, and the simultaneous presentation of CSA and CSB with the US omitted. Subsequent pairing of CSB alone with the US results in a lower rate of CR acquisition relative to animals that did not experience the compound stimulus. CSB was previously learned as a reliable predictor that the US will not occur.

In the above two examples, the two CSs had identical temporal properties. Other protocols show the effects of presenting compound stimuli that have different temporal properties (serial compound stimuli) (Figure 16.1, bottom). As mentioned earlier, acquisition of a CR is impaired in trace conditioning ($TI > 0$). However, if another CS is presented during the TI, the acquisition of the CR in response to the first CS is facilitated (*facilitation by intervening stimulus*). A related protocol results in *higher-order conditioning*, in which a CR is first acquired in response to CSB. Then, if CSA is presented prior to CSB, a CR is acquired in response to CSA. In a sense, CSB plays the role of reinforcer.

In the *primacy effect*, CSA and CSB overlap in time. The offset time of each is the same and immediately precedes the US, but the onset time of CSA is earlier than that of CSB (Figure 16.1, bottom). Because CSB has a shorter ISI than CSA, one may expect that a CR would be elicited more strongly in response to CSB alone than to CSA alone. However, the presence of CSA actually results in a decrease in the strength of the CR in response to CSB alone. More surprising is a prediction first discussed in Sutton and Barto (1981). They presented a model of classical conditioning that was first trained with CSB paired with the US (delay conditioning with a short ISI), and then CSA (with an earlier onset time) was presented as well. Even though the strength of the association between CSB and the response—which represents the predictive qualities the CS—had reached its asymptotic level, it decreased when CSA was presented. Such a finding is seemingly incongruous with the effects

of the ISI and the phenomenon of blocking. Sutton and Barto (1987) replicated this result, and Kehoe et al (1987) confirmed this prediction experimentally.

16.2.2 Theory

The blocking effect suggests that learning occurs when the unexpected happens (Kamin, 1969) as opposed to when two things are correlated. This idea led to the development of the famous *Rescorla-Wagner* model of classical conditioning (Rescorla and Wagner, 1972), in which the presence of a US during a trial is predicted by the sum of associative strengths between each CS present during the trial and the US. Changes in associative strengths depend on the accuracy of the prediction. For every CS i present during a trial:

$$\Delta w(i) = \alpha \left(r - \sum_i w(i)x(i) \right),$$

where $w(i)$ is the associative strength between CS i and the US, $x(i) = 1$ if CS i is present and 0 otherwise, r is the maximum amount of conditioning the US can produce (analogous to the “magnitude” of the US), and α is a step-size parameter. (Note that this notation differs from the original version.) If the US is perfectly-predicted (i.e., if $r - \sum_i w(i)x(i) = 0$), the associative strength of another CS subsequently added (with an initial associative strength of zero) will not increase. This influential model captures several features of classical conditioning (e.g., blocking). Also, as first noted in Sutton and Barto (1981), it is similar to the independently-developed *Widrow-Hoff* learning rule (Widrow and Hoff, 1960), showing the importance of prediction-derived learning.

The Rescorla-Wagner model is a *trial level* account of classical conditioning in that learning occurs from trial to trial as opposed to at each time point within a trial. It cannot account for the effects that temporal properties of stimuli have on learning. In addition, Sutton and Barto (1987) point out that animal learning processes may not incorporate mechanisms that are dependent on the concept of the trial, which is essentially a convenient way to segregate events.

Temporal difference (TD) models (Sutton, 1988; Sutton and Barto, 1998), which have foundations in models of classical conditioning (Sutton and Barto, 1981; Barto and Sutton, 1982; Sutton and Barto, 1987), were developed in part to do prediction learning on a *real-time* level. In the following TD model of classical conditioning (Sutton and Barto, 1987) (using notation that is similar to the equation above), let r_t be the presence (and magnitude) of the US at time step t , $x_t(i)$ be 1 if CS i is present at time t and 0 otherwise, and $w_t(i)$ be the associative strength between CS i and the US at time t . At each time point,

$$w_{t+1}(i) = w_t(i) + \alpha \left(r_t + \gamma \left[\sum_i w_t(i)x_t(i) \right]^+ - \left[\sum_i w_t(i)x_{t-1}(i) \right]^+ \right) \bar{x}_t(i),$$

where γ is the familiar temporal discount factor, $[y]^+$ returns zero if $y < 0$, and $\bar{x}_t(i)$ is an eligibility trace, e.g., $\beta\bar{x}_{t-1}(i) + (1 - \beta)x_{t-1}(i)$, where $0 \leq \beta < 1$.

At each time point, weights are adjusted to minimize the difference between $r_t + \gamma[\sum_i w_t(i)x_t(i)]^+$ and $[\sum_i w_t(i)x_{t-1}(i)]^+$ (i.e., the *temporal difference error*). These are temporally successive predictions of the same quantity: upcoming USs (more precisely, $\sum_{k=0}^{\infty} \gamma^k r_{t+k}$). The former prediction incorporates more recent information (r_t and $x_t(i)$) and serves as a target for the latter, which uses information from an earlier time point ($x_{t-1}(i)$). The eligibility trace (which restricts modification to associations of CSs that were recently present), discount factor, and explicit dependence on time allow the model to capture the effect on learning of temporal relationships among stimuli within a trial. Also, because the prediction at time t trains the prediction at time $t - 1$, the model accounts for higher order conditioning.

16.2.3 Summary and Additional Considerations

Classical conditioning reveals behavior related to an animal's ability to predict that a US will follow a CS. The prediction is thought to arise from the strengthening of an association between the CS and the US. Associations are strengthened when there is a prediction error (i.e., the animal does not already predict that the US will follow the CS), there is contingency (the US follows the CS most of the time), and there is contiguity (the US occurs shortly after the CS) (Schultz, 2006).

Methods used in TD models were developed in large part to capture the types of behaviors observed in animals engaged in classical conditioning experiments. Whereas other accounts of prediction use only the actual outcome (e.g., the US) as a training signal, TD models use the difference between temporally successive predictions of the outcome (the TD error) as a training signal. Besides providing for a better account of animal behavior, such bootstrapping has proven to be a powerful computational technique (Sutton, 1988).

Additional experiments in classical conditioning include characterizing how the form of the CR changes depending on contingencies between the CS and US and how brain mechanisms mediate such behavior. Computational accounts devised to explain behavior based on neural mechanisms further increase our understanding of how animals learn to predict (Gluck, 2008; Mirolli et al, 2010; Moore and Choi, 1997; Brandon et al, 2002).

Finally, the UR and CR can be considered *Pavlovian actions* in that the animal does not develop or choose to execute them. Rather, the US is a salient stimulus that causes the animal to emit the UR. The CR is not learned, but arises from the learned association between the stimulus (CS) and the eventual outcome (US). Because the animal has very little control over its environment and behavior, classical conditioning allows us to focus on prediction rather than decision-making. Of course, one of the benefits of prediction is that it allows us to better control what we experience. Such control is the focus of the next section.

16.3 Operant (or Instrumental) Conditioning

Classical conditioning experiments present the animal with a salient stimulus (the US) contingent on another stimulus (the CS) regardless of the animal's behavior. Operant conditioning experiments present a salient stimulus (usually a "reward" such as food) contingent on specific actions executed by the animal (Thorndike, 1911; Skinner, 1938). (The animal is thought of as an instrument that operates on the environment.) Thorndike's basic experimental protocol described at the beginning of this chapter forms the foundation of most experiments described in this section.

16.3.1 Behavior

The simplest protocols use single-action tasks such as a rat pressing the one lever or a pigeon pecking at the one key available to it. Behavior is usually described as the "strength" of the action, measured by how quickly it is initiated, how quickly it is executed, or likelihood of execution. Basic protocols and behavior are analogous to those of classical conditioning. During acquisition (action is followed by reward), action strength increases, and the reward is referred to as the reinforcer. During extinction (reward is omitted after the acquisition phase), action strength decreases. The rates of action strength increase and decrease also serve as measures of learning.

Learning depends on several factors that can be manipulated by the experimenter. In most experiments, the animal is put into a deprived state before acquisition (e.g., it's hungry if the reward is food). Action strength increases at a faster rate with deprivation; hence, deprivation is said to increase the animal's "drive" or "motivation." Other factors commonly studied include the magnitude of reward (e.g., volume of food, where an increase results in an increase in learning) and delay between action execution and reward delivery (where an increase results in a decrease).

Factors that affect learning in single-action tasks also affect selection, or decision-making, in *free choice* tasks in which more than one action is available (e.g., there are two levers). The different actions lead to outcomes with different characteristics, and the strength of an action relative to others is usually measured by relative likelihood (i.e., choice distribution). Unsurprisingly, animals more frequently choose the action that leads to a reward of greater magnitude and/or shorter delay.

We can quantify the effects of one factor in terms of another by examining choice distribution. For example, suppose action A leads to a reward of a constant magnitude and delay and action B leads to an immediate reward. By determining how much we must decrease the magnitude of the immediate reward so that the animal shows no preference between the two actions, we can describe a temporal discount function. Although most accounts in RL use an exponential discount function, behavioral studies support a hyperbolic form (Green and Myerson, 2004).

Probability of reward delivery is another factor that affects choice distribution. In some tasks where different actions lead to rewards that are delivered with different probabilities, humans and some animals display *probability matching*, where

the choice distribution is similar to the relative probabilities that each action will be reinforced (Siegel and Goldstein, 1959; Shanks et al, 2002). Such a strategy is clearly suboptimal if the overall goal is to maximize total reward received. Some studies suggest that probability matching may be due in part to factors such as the small number of actions that are considered in most experiments or ambiguous task instructions (i.e., participants may be attempting to achieve some goal other than reward maximization) (Shanks et al, 2002; Gardner, 1958; Goodnow, 1955).

Naive animals usually do not execute the specific action(s) the experimenter wishes to examine; the animal must be “taught” with methods drawn from the basic results outlined above and from classical conditioning. For example, in order to draw a pigeon’s attention to a key to be pecked, the experimenter may first simply illuminate the key and then deliver the food, independent of the pigeon’s behavior. The pigeon naturally pecks at the food and, with repeated pairings, pecks at the key itself (*autoshaping*, Brown and Jenkins 1968). Although such Pavlovian actions can be exploited to guide the animal towards some behaviors, they can hinder the animal in learning other behaviors (Dayan et al, 2006).

If the movements that compose an action can vary to some degree, a procedure called *shaping* can be used to teach an animal to execute a specific movement by gradually changing the range of movements that elicit a reward (Eckerman et al, 1980). For example, to teach a pigeon to peck at a particular location in space, the experimenter defines a large imaginary sphere around that location. When the pigeon happens to peck within that sphere, food is delivered. When the pigeon consistently pecks within that sphere, the experimenter decreases the radius, and the pigeon receives food only when it happens to peck within the smaller sphere. This process continues until an acceptable level of precision is reached.

Shaping and autoshaping modify the movements that compose a single action. Some behaviors are better described as a chain of several actions executed sequentially. To train animals, experimenters exploit higher-order conditioning (or *conditioned reinforcement*): as with classical conditioning, previously neutral stimuli can take on the reinforcing properties of a reinforcer with which they were paired. In *backward chaining*, the animal is first trained to execute the last action in a chain. Then it is trained to execute the second to last action, after which it is in the state from which it can execute the previously acquired action (Richardson and Warzak, 1981). This process, in which states from which the animal can execute a previously learned action act as a reinforcers, continues until the entire sequence is learned.

16.3.2 Theory

To account for the behavior he had observed in his experiments, Thorndike devised his famous Law of Effect:

Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to

recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond. (Chapter 5, page 244 of Thorndike 1911.)

Learning occurs only with experience: actually executing the action and evaluating the outcome. According to Thorndike, action strength is due to the strength of an association between the response (or action) and the situation (state) from which it was executed. The basic concepts described in the Law of Effect are also used in many RL algorithms. In particular, action strength is analogous to *action value*, $Q(s,a)$, which changes according to the consequences of the action and determines behavior in many RL schemes.

This type of action generation, where the action is elicited by the current state, is sometimes thought of as being due to a *stimulus-response* (SR) association. Though the term is a bit controversial, I use it here because it is used in many neuroscience accounts of behavior (e.g., Yin et al 2008) and it emphasizes the idea that behavior is due to associations between available actions and the current state. In contrast, actions may be chosen based on explicit predictions of their outcomes (discussed in the next subsection).

Thorndike conducted his experiments in part to address questions that arose from Charles Darwin's Theory of Evolution: do humans and animals have similar mental faculties? His Law of Effect provides a mechanistic account that, coupled with variability (exploration), can explain even complicated behaviors. The SR association, despite its simplicity, plays a role in several early psychological theories of behavior (e.g., Thorndike 1911; Hull 1943; Watson 1914). New SR associations can be formed from behavior generated by previously-learned ones, and a complicated "response" can be composed of previously-learned simple responses. (Such concepts are used in hierarchical methods as well, Grupen and Huber 2005; Barto and Mahadevan 2003 and see also Chapter 9.) This view is in agreement with Evolution: simple processes of which animals are capable explain some human behavior as well. (In Chapter 6 of his book, Thorndike suggests that thought and reason are human qualities that arise from our superior ability to learn associations.)

As with animals, it may be very difficult for a naive artificial agent to learn a specific behavior. Training procedures developed by experimentalists, such as backward chaining and shaping, can be used to aid artificial agents as well (Konidaris and Barto, 2009; Selfridge et al, 1985; Ng et al, 1999). A type of shaping also occurs when reinforcement methods are used to address the structural credit assignment problem (Barto, 1985): when an "action" is composed of multiple elements that can each be modified, exactly *what* did an agent just do that led to the reward?

Psychological theories that primarily use concepts similar to the SR association represent a view in which behavior is accounted for only by variables that can be directly-observed (e.g., the situation and the response). Taken to the extreme, it is controversial and cannot explain all behavior. The next subsection discusses experiments that show that some behavior is better explained by accounts that do allow for variables that cannot be directly-observed.

16.3.3 Model-Based Versus Model-Free Control

The idea that an action is elicited from an SR association is a *model-free* account of behavior—an explicit prediction of the outcome of the action plays no role in its execution. However, the results of other experiments led to theories in which an action is elicited from an explicit prediction of its outcome—*model-based* accounts. For example, rats that had prior exposure to a maze in the absence of any reward quickly learned the same route to a reward later presented as rats that had been trained with the reward all along. This behavior, and that from other maze experiments, led the American psychologist Edward Tolman to suggest that rats form models of the environment which specify relations between states and between actions and their outcomes (Tolman, 1948). Whereas an SR association is strengthened in model-free accounts, an association between an action and its predicted outcome is strengthened in many types of model-based accounts.

To more directly determine if an explicit representation of outcome had any effect on behavior, *goal devaluation* (or *revaluation*) procedures have been devised (Dickinson, 1985; Dickinson and Balleine, 1994; Balleine et al, 2009). In a typical experiment, an animal is trained to perform some operant task and then, in a separate situation, its motivation for the reward is changed (e.g., by making it ill just after it eats the food). The animal is then placed in the original task environment during an extinction test. (To prevent new learning, it is not given the outcome.) If behavior is different than that of animals that have not undergone the devaluation procedure (e.g., if action strength during extinction declines at a faster rate), we can infer that it is controlled at least in part by some representation of its outcome.

This is indeed the case for animals that have not had extensive training, suggesting that they used model-based mechanisms (in many cases, the animal does not execute the action even on the first trial after devaluation). However, the behavior of animals that have had extensive training is not affected by devaluation. Such behavior is considered habitual (Dickinson, 1985), and it is better accounted for by model-free mechanisms. Dickinson (1985) suggests that behavior is the result of one of two processes: as the animal learns the task, there is a high correlation between rate of behavior and rate of reinforcer delivery, and this correlation strengthens action-outcome (AO) associations. Meanwhile, SR associations are being learned as well. However, with extensive training, behavior rate plateaus and, because its variance is very small, its correlation with reinforcer delivery is much weaker. Thus the AO association is decreased and SR associations dominate control.

The availability of multiple control mechanisms has functional advantages. Because the outcome is explicitly represented, a model-based controller does not need much experience in order to make appropriate decisions once it has encountered the reward. Consequently, it can adapt quickly—even immediately—if the reward structure changes (i.e., it's flexible). However, making decisions based on predictions generated by a model has high computational and representational demands. A model-free controller requires fewer resources but much more experience in order to develop SR associations so that appropriate decisions are made. Such requirements make them less flexible.

Further insights can be drawn from computational models of decision-making that use multiple controllers. In a particularly relevant account (Daw et al, 2005), model-based and model-free RL algorithms were used to select actions in a simulated goal devaluation task. Because the task changes when the goal is devalued, a measure of uncertainty was incorporated into the algorithms. At each state, the action suggested by the controller with the lower uncertainty was executed. Because the model-free controller required much experience, the model-based controller dominated control early on. However, because the model-based controller relied on multiple predictions to select actions, the minimum uncertainty it could achieve was higher than that of the model-free controller, and the model-free controller dominated later. The multiple controller scheme accounted for much of the behavior observed experimentally and provided computationally-grounded explanations for how such behavior is developed. Other types of multiple controller models suggest a more passive arbitration scheme, where the simpler controller simply selects an action faster if it is sufficiently trained (Ashby et al, 2007; Shah and Barto, 2009; Shah, 2008). The simpler mechanisms may be useful in the construction of hierarchical behavior such as skills or “chunks” of actions (Shah, 2008).

Goal devaluation experiments provide elegant behavioral evidence that animals use different control mechanisms at different stages in learning. Equally elegant experiments show that different brain systems are involved in mediating the different control mechanisms (Balleine et al, 2009). Computational models inspired by these results use learning and control mechanisms that are thought to occur in the corresponding brain areas. These connections are discussed further in Section 5.3.

16.3.4 Summary and Additional Considerations

Operant conditioning shows us that the consequences of behavior can be used to train animals to develop specific actions and adjust the strength—including likelihood of execution—of those specific actions. Such actions may be referred to as *operant actions* to distinguish them from Pavlovian actions. Psychological theories that explain such development have much in common with RL: that a scalar reinforcement signal strengthens an association between stimulus and response (SR, consistent with model-free mechanisms) or action and outcome (AO, model-based).

Several other types of processes gleaned from experiments in animal behavior may be of interest to the reader. As discussed earlier, an animal’s motivational state affects its behavior. Early attempts to explain this effect have led to the drive theory of Hull, where *any* learned SR association is strengthened when the animal is more motivated (Hull, 1943), and the incentive value theory of Tolman, where the strength of an action depends on the motivational state of the animal when it learned that action, even if it is no longer motivated (Tolman, 1949). These ideas are further explored experimentally (Dickinson and Balleine, 1994; Pompilio and Kacelnik, 2005) and within an RL framework (Niv et al, 2006b).

The experiments described in this section deliver a reward when an action is executed, but other experiments examine behavior when the action results in an aversive outcome (such as a shock) or the absence of a salient outcome. Other types of manipulations include delivering the outcome only after some number of actions have been executed or amount of time has elapsed since the last delivery (Ferster and Skinner, 1957). These manipulations result in interesting behavior (e.g., the *matching law*, Herrnstein 1961) that may further reveal the underlying processes of learning and control in animals (Staddon and Cerutti, 2003).

In general, we can think of an animal as doing what we develop RL agents to do: modify its behavior so as to maximize satisfaction using only a coarse evaluation of the consequences of behavior as feedback. Although much can be learned by examining animal behavior, we must turn to neuroscience in order to better understand the neural processes that govern such behavior. The next two sections describe evidence suggesting that animals use learning processes remarkably similar to those used in many RL algorithms.

16.4 Dopamine

Although different types of rewards could have similar effects on behavior, it was unknown if similar or disparate neural mechanisms mediate those effects. In the early 1950s, it was discovered that if an action led to electrical stimulation applied (via an electrode) to the brain, action strength would increase (Olds and Milner, 1954). This technique, known as *intracranial self-stimulation* (ICSS), showed greater effects when the electrodes were strongly stimulating the projections of neurons that release dopamine (DA) from their axon terminals. DA neurons are mostly located in the substantia nigra pars compacta (SNpc, also called the A9 group) (a part of the basal ganglia, BG) and the neighboring ventral tegmental area (VTA, group A10) (Björklund and Dunnett, 2007). As described in the next section, many of the areas DA neurons project to are involved with decision-making and movement. ICSS shows that behavior may be modified according to a global signal communicated by DA neurons. Original interpretations suggested, quite reasonably, that DA directly signals the occurrence of a reward. Subsequent research, some of which is described in this section, shows that DA plays a more sophisticated role (Wise, 2004; Schultz, 2007; Bromberg-Martin et al, 2010; Montague et al, 2004).

16.4.1 Dopamine as a Reward Prediction Error

To better characterize the role of DA in behavior, researchers recorded the activity of single neurons from VTA and SNpc in awake behaving animals. DA neurons exhibit low baseline firing-rates (< 10 Hz) with short (*phasic*) bursts of firing (henceforth referred to as “*the DA burst*,” and the magnitude of the burst refers to the frequency

of firing within a burst). Early studies showed that a DA burst occurred in response to sensory events that were task-related, intense, or surprising (Miller et al, 1981; Schultz, 1986; Horvitz, 2000).

In one of these most important studies linking RL to brain processes, Ljungberg et al (1992) recorded from DA neurons in monkeys while the monkeys learned to reach for a lever when a light was presented, after which they received some juice. Initially, a DA burst occurred in response only to juice delivery. As the task was learned, a DA burst occurred at both the light and juice delivery, and later only to the light (and *not* juice delivery). Finally, after about 30,000 trials (over many days), the DA burst even in response to the light declined by a large amount (perhaps due to a lack of attention or motivation, Ljungberg et al 1992). Similarly, Schultz et al (1993) showed that as monkeys learned a more complicated operant conditioning task, the DA burst moved from the time of juice delivery to the time of the stimuli that indicated that the monkey should execute the action. If the monkey executed the wrong action, DA neuron activity at the time of the expected juice delivery decreased from baseline. Also, juice delivered before its expected time resulted in a DA burst; the omission of expected juice (even if the monkey behaved correctly) resulted in a decrease in activity from baseline; and the DA burst at the time of juice delivery gradually decreased as the monkey learned the task (Schultz et al, 1997; Hollerman and Schultz, 1998).

The progression of DA neuron activity over the course of learning did not correlate with variables that were directly-manipulated in the experiment, but it caught the attention of those familiar with RL (Barto, 1995). As noted by Houk et al (1995), there “is a remarkable similarity between the discharge properties of DA neurons and the effective reinforcement signal generated by a TD algorithm...” (page 256). Montague et al 1996 hypothesized that “the fluctuating delivery of dopamine from the VTA to cortical and subcortical target structures in part delivers information about prediction errors between the expected amount of reward and the actual reward” (page 1944). Schultz et al (1997) discuss in more detail the relationship between their experiments, TD, and psychological learning theories (Schultz, 2010, 2006, 1998). The importance of these similarities cannot be overstated: a fundamental learning signal developed years earlier within the RL framework appears to be represented—almost exactly—in the activity of DA neurons recorded during learning tasks.

Several studies suggest that the DA burst is, like the TD error, influenced by the prediction properties of a stimulus, e.g., as in blocking (Waelti et al, 2001) and conditioned inhibition (Tobler et al, 2003). When trained monkeys were faced with stimuli that predict the probability of reward delivery, the magnitude of the DA burst at the time of stimuli presentation increased with likelihood; that at the time of delivered reward decreased; and DA neuron activity at the time of an omitted reward decreased to a larger extent (Fiorillo et al, 2003) (but see Niv et al 2005). When reward magnitude was drawn from a probability distribution, the DA burst at the time of reward delivery reflected the difference between delivered and expected

magnitude (Tobler et al, 2005). Other influences on the DA burst include motivation (Satoh et al, 2003), delay of reward delivery, (Kobayashi and Schultz, 2008), and history (if reward delivery was a function of past events) (Nakahara et al, 2004; Bayer and Glimcher, 2005).

In a task that examined the role of DA neuron activity in decision-making (Morris et al, 2006), monkeys were presented with visual stimuli indicating probability of reward delivery. If only one stimulus was presented, DA neuron activity at the time of the stimulus presentation was proportional to the expected value. If two stimuli were presented, and the monkey could choose between them, DA neuron activity was proportional to the expected value of the eventual choice rather than representing some combination of the expected values of the available choices. The authors suggest that these results indicate that DA neuron activity reflects the perceived value of a chosen (by some other process) action, in agreement with a SARSA learning scheme (Niv et al, 2006a) (though the results of other experiments support a Q-learning scheme, Roesch et al 2007).

16.4.2 Dopamine as a General Reinforcement Signal

The studies reviewed above provide compelling evidence that the DA burst reflects a reward prediction error very similar to the TD error of RL. Such an interpretation is attractive because we can then draw upon the rich computational framework of RL to analyze such activity. However, other studies and interpretations suggest that DA acts as a general reinforcer of behavior, but perhaps not just to maximize reward.

The *incentive salience* theory (Berridge and Robinson, 1998; Berridge, 2007; Berridge et al, 2009) separates “wanting” (a behavioral bias) from “liking” (hedonic feelings). Experiments using pharmacological manipulations suggest that opioid—not DA—systems mediate facial expressions associated with pleasure (Berridge and Robinson, 1998). DA could increase action strength without increasing such measures (Tindell et al, 2005; Wyvill and Berridge, 2000), and DA-deficient animals could learn to prefer pleasurable stimuli over neutral ones (Cannon and Palmiter, 2003). The separation offers an explanation for some experimental results and irrational behaviors (Wyvill and Berridge, 2000).

Redgrave et al (2008) argue that the latency of the DA burst, < 100 ms after stimulus presentation in many cases, is too short and uniform (across stimuli and species) to be based on identification—and hence predicted value—of the stimulus. Rather, the DA burst may be due to projections from entirely subcortical pathways that respond quickly—faster than DA neurons—to coarse perceptions that indicate that something has happened, but not what it was (Dommett et al, 2005). More recent experimental results provide evidence that additional phasic DA neuron activity that occurs with a longer latency (< 200 ms) (Joshua et al, 2009; Bromberg-Martin et al, 2010; Nomoto et al, 2010) may be due to early cortical processing and does provide some reward-related information. Very early (< 100 ms) DA activity may signal a sensory prediction error (e.g., Horvitz 2000) that biases the animal to repeat

movements so as to determine what it had done, if anything, to cause the sensory event (Redgrave et al, 2008; Redgrave and Gurney, 2006). Later DA activity may be recruited to bias the animal to repeat movements in order to maximize reward.

16.4.3 Summary and Additional Considerations

The experiments described this section show that DA acts not as a “reward detector,” but rather as a learning signal that reinforces behavior. Pharmacological treatments that manipulate the effectiveness of DA further support this idea. For example, in humans that were given DA agonists (which increase the effectiveness of DA on target neurons) while performing a task, there was an increase in both learning and a representation of the TD error in the striatum (a brain area targeted by DA neurons) (Pessiglione et al, 2006). DA antagonists (which decrease the effectiveness of DA) had the opposite effect.

There are a number of interesting issues that I have not discussed but deserve mention. Exactly how the DA burst is shaped is a matter of some debate. Theories based on projections to and from DA neurons suggest that they are actively suppressed when a predicted reward is delivered (Hazy et al, 2010; Houk et al, 1995). Also, because baseline DA neuron activity is low, aversive outcomes or omitted rewards cannot be represented in the same way as delivered rewards. Theories based on stimulus representation (Ludvig et al, 2008; Daw et al, 2006a), other neurotransmitter systems (Daw et al, 2002; Phelps and LeDoux, 2005; Wräse et al, 2007; Doya, 2008), and/or learning rules (Frank, 2005; Frank et al, 2004) address this issue. While this section focused on phasic DA neuron activity, research is also examining the effect that long-term (*tonic*) DA neuron activity has on learning and behavior (Schultz, 2007; Daw and Touretzky, 2002). Finally, recent experimental evidence suggests that the behavior of DA neurons across anatomical locations may not be as uniform as suggested in this section (Haber, 2003; Wickens et al, 2007).

While several interpretations of how and to what end DA affects behavior have been put forth, of most interest to readers of this chapter is the idea that the DA burst represents a signal very similar to the TD error. To determine if that signal is used in the brain in ways similar to how RL algorithms is it, we must examine target structures of DA neurons.

16.5 The Basal Ganglia

Most DA neuron projections terminate in frontal cortex and the basal ganglia (BG), areas of the brain that are involved in the control of movement, decision-making, and other cognitive processes. Because DA projections to the BG are particularly dense relative to projections in frontal cortex, this section focuses on the BG. What follows is a brief (and necessarily incomplete) overview of the BG and its role in learning and control.

16.5.1 Overview of the Basal Ganglia

The BG are a set of interconnected subcortical structures located near the thalamus. Scientists first connected their function with voluntary movement in the early twentieth century when post-mortem analysis showed that a part of the BG was damaged in Parkinson's disease patients. Subsequent research has revealed a basic understanding of their function in terms of movement (Mink, 1996), and research over the past few decades show that they mediate learning and cognitive functions as well (Packard and Knowlton, 2002; Graybiel, 2005).

A part of the BG called the striatum receives projections from DA neurons and excitatory projections from most areas of cortex and thalamus. A striatal neuron receives a large number of weak inputs from many cortical neurons, suggesting that striatal neurons implement a form of pattern recognition (Houk and Wise, 1995; Wilson, 2004). Striatal neurons send inhibitory projections to the internal segment of the globus pallidus (GPi), the neurons of which are tonically-active and send inhibitory projections to brain stem and thalamus. Thus, excitation of striatal neurons results in a disinhibition of neurons targeted by GPi neurons. In the case in which activation of neurons targeted by the GPi elicits movements, their disinhibition increases the likelihood that those movements will be executed.

On an abstract level, we can think of pattern recognition at striatal neurons as analogous to the detection of state as used in RL, and the resulting disinhibition of the targets of GPi neurons as analogous to the selection of actions. Corticostriatal synapses are subject to DA-dependent plasticity (Wickens, 2009; Calabresi et al, 2007), e.g., a learning rule roughly approximated by the product of the activity of the striatal neuron and the activities of the cortical and DA neurons that project to it. Thus, the DA burst (e.g., representing the TD error) can modify the activation of striatal neurons that respond to a particular state according to the consequences of the resulting action. In other words, the BG possess characteristics that enable them to modify the selection of actions through mechanisms similar to those used in RL (Barto, 1995; Doya, 1999; Daw and Doya, 2006; Doll and Frank, 2009; Graybiel, 2005; Joel et al, 2002; Wickens et al, 2007).

Additional pathways within the BG (which are not described here) impart a functionality to the BG useful for behavioral control. For example, actions can be actively facilitated or inhibited, possibly through different learning mechanisms (Frank, 2005; Frank et al, 2004; Hikosaka, 2007). Also, intra-BG architecture appears to be well-suited to implement selection between competing actions in an optimal way (Bogacz and Gurney, 2007; Gurney et al, 2001).

Different areas of cortex—which are involved in different types of functions—project to different areas of the striatum. These pathways stay segregated to a large degree through the BG, to the thalamus, and back up to the cortex (Alexander et al, 1986; Middleton and Strick, 2002). The parallel loop structure allows the BG to affect behavior by shaping cortical activity as well as through descending projections to brain stem. The functional implications of the parallel loop structure are discussed later in this section. The next subsection describes studies that aim to

further elucidate the functions of the BG, focusing mostly on areas involved with the selection of movements and decisions.

16.5.2 Neural Activity in the Striatum

In most of the experiments described in this subsection, the activities of single neurons in the striatum were recorded while the animal was engaged in some conditioning task. As the animal learned the task, neural activity began to display task-related activity, including activity modulated by reward (Schultz et al, 2003; Hikosaka, 2007; Barnes et al, 2005).

In a particularly relevant study, Samejima et al (2005) recorded from dorsal striatum (where dorsal means toward the top of the head) of monkeys engaged in a two-action free choice task. Each action led to a large reward (a large volume of juice) with some probability that was held constant over a block of trials, and a smaller reward (small volume) the rest of the time. For example, in one block of trials, the probability that action A led to the large reward was 0.5 and that of action B was 0.9, while in another block the probabilities were, respectively, 0.5 and 0.1. Such a design dissociates the absolute and relative action values (the expected reward for executing the action): the value of action A is the same in both blocks, but it is higher than that of action B in the first block and lower in the second. Choices during a block were distributed between the two actions, with a preference for the more valuable one.

The recorded activity of about one third of the neurons during a block covaried with the value of one of the actions (a lesser proportion covaried with other aspects such as the difference in action values or the eventual choice). In addition, modelling techniques were used to estimate action values online based on experience, i.e., past actions and rewards, and to predict choice behavior based on those estimated action values. Many neurons were found whose activities covaried with estimated action value, and the predicted choice distribution agreed with the observed distribution.

The temporal profile of neural activity within a trial yields further insights. Lau and Glimcher (2008) showed that dorsal striatal neurons that encoded the values of available actions were more active before action execution, and those that encoded the value of the chosen action were more active after action execution. The results of these and other studies (Balleine et al, 2007; Kim et al, 2009) suggest that action values of some form are represented in dorsal striatum and that such representations participate in action selection and evaluation (though some lesion studies suggest that dorsal striatum may not be needed for learning such values, Atallah et al 2007).

Analyses described in Samejima et al (2005) illustrate an approach that has been growing more prominent over the past decade: to correlate neural activity not only with variables that can be directly-observed (e.g., reward delivery or choice), but also with variables thought to participate in learning and control according to theories and computational models (e.g., expected value) (Corrado and Doya, 2007; Daw and Doya, 2006; Niv, 2009). This approach is especially useful in analyzing data

derived from functional magnetic resonance imaging (fMRI) methods (Gläscher and O'Doherty, 2010; Montague et al, 2006; Haruno and Kawato, 2006), where the activity of many brain areas can be recorded simultaneously, including in humans engaged in complex cognitive tasks. Note that the precise relationship between the measured signal (volume of oxygenated blood) and neural activity is not known and the signal has a low temporal and spatial resolution relative to single neuron recordings. That being said, analyses of the abundance of data can give us a better idea of the overall interactions between brain areas.

Using fMRI, O'Doherty et al (2004) showed that, in a task in which the human participant must choose an action from a set, dorsolateral striatum and ventral striatum (where ventral means toward the bottom) exhibited TD error-like signals, while in a task in which the participant had no choice, only ventral striatum exhibited such a signal. These results suggest that dorsal and ventral striatum implement functions analogous to the Actor and Critic (see also Barto 1995; Joel et al 2002; Montague et al 2006), respectively, in the Actor-Critic architecture (Barto et al, 1983).

Recordings from single neurons in the ventral striatum of animals support this suggestion to some degree. Information which can be used to evaluate behavior, such as context (or state), some types of actions, and outcome, appear to be represented in ventral striatum (Ito and Doya, 2009; Roesch et al, 2009; Kim et al, 2009). Roughly speaking, while dorsal striatum is more concerned with actions in general, ventral striatum may participate in assigning value to stimuli, but it may also participate in controlling some types of actions and play a more complicated role in behavior (Yin et al, 2008; Humphries and Prescott, 2010; Nicola, 2007).

16.5.3 Cortico-basal Ganglia-thalamic Loops

As described earlier, pathways from cortex to the BG to thalamus and back up to cortex form parallel segregated loops (Alexander et al, 1986; Middleton and Strick, 2002). The basic functionality of the BG within a loop—RL-mediated selection or biasing—is thought to apply to information that is represented by neural activity within that loop (Wickens et al, 2007; Samejima and Doya, 2007; Graybiel et al, 1994; Houk et al, 2007; Yin et al, 2008; Haruno and Kawato, 2006; Redgrave et al, 2010; Packard and Knowlton, 2002; Cohen and Frank, 2009). Also, because the different cortical areas are involved in different types of functions, the loop structure allows the BG to affect behavior through different mechanisms (e.g., those that govern behavior in classical and operant conditioning, see Sections 2.4 and 3.3). In an interpretation similar to that put forth by Yin et al (2008), these are: 1) operant actions learned through model-free mechanisms (i.e., generated by a stimulus-response, SR, association); 2) operant actions learned through model-based mechanisms (action-outcome, AO, association); or 3) the learning of stimulus-outcome, SO, associations, which can result in the execution of Pavlovian actions.

A model-free mechanism is thought to be implemented in a loop involving the dorsolateral striatum (DLS, also called the putamen in primates). The DLS receives

cortical projections mainly from primary sensory, motor, and premotor cortices (which will be referred to collectively as sensorimotor cortices, or SMC), providing the DLS with basic sensory and movement information.

A model-based mechanism is thought to be implemented in a loop involving the dorsomedial striatum (DMS, also called the caudate), which receives cortical projections mainly from prefrontal cortex (PFC). The PFC is on the front part of the cortex and has reciprocal connections with many other cortical areas that mediate abstract representations of sensations and movement. PFC neurons exhibit sustained activity (working memory, Goldman-Rakic 1995) that allows them to temporarily store information. RL mechanisms mediated by the DMS may determine which past stimuli should be represented by sustained activity (O'Reilly and Frank, 2006). Also, the PFC is thought to participate in the construction of a model of the environment (Gläscher et al, 2010), allowing it to store predicted future events as well (Mushiake et al, 2006; Matsumoto et al, 2003). Thus, the PFC can affect behavior through a planning process and even override behavior suggested by other brain areas (Miller and Cohen, 2001; Tanji and Hoshi, 2008).

The assignment of value to previously neutral stimuli may be implemented in a loop involving the ventral striatum (VS, also called the nucleus accumbens), which receives cortical projections mainly from the orbitofrontal cortex (OFC, the underside part of the PFC that is just behind the forehead.) The VS and the OFC have connections with limbic areas, such as the amygdala and hypothalamus. These structures, and VS and OFC, have been implicated in the processing of emotion, motivations, and reward (Wallis, 2007; Cardinal et al, 2002; Mirolli et al, 2010).

Most theories of brain function that focus on the loop structure share elements of the interpretation of Yin et al (2008) (though of course there are some differences) (Samejima and Doya, 2007; Haruno and Kawato, 2006; Daw et al, 2005; Balleine et al, 2009; Ashby et al, 2010; Balleine and O'Doherty, 2010; Pennartz et al, 2009; Houk et al, 2007; Wickens et al, 2007; Redgrave et al, 2010; Mirolli et al, 2010; Packard and Knowlton, 2002; Cohen and Frank, 2009). Common to all is the idea that different loops implement different mechanisms that are useful for the types of learning and control described in this chapter. Similar to how behavioral studies suggest that different mechanisms dominate control at different points in learning (e.g., control is transferred from model-based to model-free mechanisms, as discussed in Section 3.3), neuroscience studies suggest that brain structures associated with different loops dominate control at different points in learning (previous references and Doyon et al 2009; Poldrack et al 2005).

For example, in humans learning to perform a sequence of movements, the PFC (model-based mechanisms) dominates brain activity (as measured by fMRI) early in learning while the striatum and SMC (model-free) dominates activity later (Doyon et al, 2009). These results can be interpreted to suggest that decision-making mechanisms during model-based control lie predominantly within the PFC, while those during model-free control lie predominantly at the cortico-striatal synapses to the DLS. That is, control is transferred from cortical to BG selection mechanisms (Daw et al, 2005; Niv et al, 2006b; Shah, 2008; Shah and Barto, 2009), in rough agreement with experimental studies that suggest that the BG play a large role in encoding

motor skills and habitual behavior (Graybiel, 2008; Pennartz et al, 2009; Aldridge and Berridge, 1998). However, other theories and experimental results suggest that control is transferred in the opposite direction: the BG mediate trial and error learning early on (Pasupathy and Miller, 2005; Packard and Knowlton, 2002), but cortical areas mediate habitual or skilled behavior (Ashby et al, 2007, 2010; Frank and Claus, 2006; Matsuzaka et al, 2007). As discussed in Ashby et al (2010), part of the discrepancy may be due to the use of different experimental methods, including tasks performed by the animal and measures used to define habitual or skilled behavior.

Finally, some research is also focusing on how control via the different loops is coordinated. Behavior generated by mechanisms in one loop can be used to train mechanisms of another, but there is also some communication between the loops (Haber, 2003; Haber et al, 2006; Pennartz et al, 2009; Yin et al, 2008; Balleine and O'Doherty, 2010; Mirolli et al, 2010; Joel and Weiner, 1994; Graybiel et al, 1994). Such communication may occur within the BG (Pennartz et al, 2009; Graybiel, 2008; Joel and Weiner, 1994; Graybiel et al, 1994) or via connections between striatum and DA neurons (some of which are also part of the BG). The latter connections are structured such that an area of the striatum projects to DA neurons that send projections back to it and also to a neighboring area of striatum (Haber, 2003). The pattern of connectivity forms a spiral where communication is predominantly in one direction, suggestive of a hierarchical organization in which learned associations within the higher-level loop are used to train the lower-level loop (Haruno and Kawato, 2006; Yin et al, 2008; Samejima and Doya, 2007). Again following an interpretation similar to that of Yin et al (2008), the OFC-VS loop (SO association) informs the PFC-DMS loop (AO), which informs the SMC-DLS loop (SR).

16.5.4 Summary and Additional Considerations

BG function is strongly affected by DA, which we can approximate as representing the TD error. The architectural and physiological properties of the BG endow it with the ability to do RL. Experimental studies show that neurons in the dorsal striatum, which communicates with motor-related and decision-making areas of the brain, represent action values. These results suggest that the BG mediate learning and control in ways similar to those used in many RL algorithms.

Studies also suggest that different cortico-basal ganglia-thalamic loops use RL mechanisms on different types of information. The parallel loop structure allows for behavior to be affected through different mechanisms, and communication between loops allows for learning in one mechanism to drive learning in another. The conceptual architecture derived from biological systems suggests a potentially advantageous way to construct hierarchical control architectures for artificial agents, both in what types of information different levels of the hierarchy represent and how the different levels communicate to each other.

Of course, pathways, brain areas, and functional mechanisms other than those described in this section influence learning and control as well. For example, DA

affects neural activity directly, striatal activity is shaped by the activities of interneurons (neurons that project to other neurons within the same structure), which change dramatically as an animal learns a task (Graybiel et al, 1994; Graybiel, 2008; Pennartz et al, 2009), and the BG also affects behavior through recurrent connections with subcortical structures (McHaffie et al, 2005). Reward-related activity in frontal cortical areas (Schultz, 2006) are shaped not only through interactions with the BG, but also by direct DA projections and interactions with other brain areas.

Computational mechanisms and considerations that were not discussed here but are commonly used in RL and machine learning have analogs in the brain as well. Psychological and neuroscientific research address topics such as behavior under uncertainty, state abstraction, game theory, exploration versus exploitation, and hierarchical behavior (Dayan and Daw, 2008; Doya, 2008; Gold and Shadlen, 2007; Seger and Miller, 2010; Glimcher and Rustichini, 2004; Daw et al, 2006b; Yu and Dayan, 2005; Wolpert, 2007; Botvinick et al, 2009; Grafton and Hamilton, 2007). Determining how brain areas contribute to observed behavior is a very difficult endeavour. The approach discussed earlier—analyzing brain activity in terms of variables used in principled computational accounts (Corrado and Doya, 2007; Daw and Doya, 2006; Niv, 2009)—is used for a variety of brain areas and accounts beyond that discussed in this section. For example, the field of neuroeconomics—which includes many ideas discussed in this chapter—investigates decision-making processes and associated brain areas in humans engaged in economic games (Glimcher and Rustichini, 2004; Glimcher, 2003).

16.6 Chapter Summary

RL is a computational formulation of learning, through interaction with the environment, to execute behavior that delivers satisfying consequences. Animals possess this ability, and RL was inspired in large part by early studies in animal behavior and the psychological theories they spawned (Sections 2 and 3). Methodological advances in biology enabled scientists to better characterize learning and control mechanisms of animals by recording the neural activity of animals engaged in learning tasks. RL provides a framework within which to analyze such activity, and it was found that some mechanisms the brain uses are remarkably similar to mechanisms used in RL algorithms (Sections 4 and 5).

In particular, the characterization of DA neuron activity in terms of the TD error (Section 4.1) has forged a stronger and more explicit communication between RL (and machine learning) and psychology and neuroscience. A number of outstanding reviews have been published recently that focus on these connections: Niv 2009; Dayan and Daw 2008; Daw and Doya 2006; Cohen 2008; Doya 2007; Maia 2009; Dayan and Niv 2008. In addition, studies in computational neuroscience show how physiological and architectural characteristics of brain areas can implement specific functions: Wörgötter and Porr 2005; Cohen and Frank 2009; Gurney et al 2004; Gurney 2009; Doya 2008; Bar-Gad et al 2003; Joel et al 2002; Hazy et al 2010;

Mirolli et al 2010. The integration of computational methods with psychology and neuroscience not only gives us a better understanding of decision-making processes, but it also presents us with new approaches to study disorders in decision-making such as mental disorders and addiction (Maia and Frank, 2011; Kishida et al, 2010; Redish et al, 2008; Redgrave et al, 2010; Graybiel, 2008; Belin et al, 2009).

Finally, more recent research in neuroscience is beginning to further unravel the mechanisms by which animals develop complex behavior. In particular, rather than use one monolithic control mechanism, animals appear to use multiple control mechanisms (Sections 3.3 and 5.3). Neuroscientific research suggests how these different mechanisms cooperate (Section 5.3). Such schemes contribute to the sophisticated behavior of which animals are capable and may serve as inspiration in our quest to construct sophisticated autonomous artificial agents.

Acknowledgements. I am grateful for comments from and discussions with Andrew Barto, Tom Stafford, Kevin Gurney, and Peter Redgrave, comments from anonymous reviewers, and financial support from the European Community's 7th Framework Programme grant 231722 (IM-CLeVeR).

References

- Aldridge, J.W., Berridge, K.C.: Coding of serial order by neostriatal neurons: a “natural action” approach to movement sequence. *The Journal of Neuroscience* 18, 2777–2787 (1998)
- Alexander, G.E., DeLong, M.R., Strick, P.L.: Parallel organization of functionally segregated circuits linking basal ganglia and cortex. *Annual Review of Neuroscience* 9, 357–381 (1986)
- Ashby, F.G., Ennis, J., Spiering, B.: A neurobiological theory of automaticity in perceptual categorization. *Psychological Review* 114, 632–656 (2007)
- Ashby, F.G., Turner, B.O., Horvitz, J.C.: Cortical and basal ganglia contributions to habit learning and automaticity. *Trends in Cognitive Sciences* 14, 208–215 (2010)
- Atallah, H.E., Lopez-Paniagua, D., Rudy, J.W., O'Reilly, R.C.: Separate neural substrates for skill learning and performance in ventral and dorsal striatum. *Nature Neuroscience* 10, 126–131 (2007)
- Balleine, B.W., O'Doherty, J.P.: Human and rodent homologies in action control: Corticostriatal determinants of goal-directed and habitual action. *Neuropsychopharmacology* 35, 48–69 (2010)
- Balleine, B.W., Delgado, M.R., Hikosaka, O.: The role of the dorsal striatum in reward and decision-making. *The Journal of Neuroscience* 27, 8161–8165 (2007)
- Balleine, B.W., Liljeholm, M., Ostlund, S.B.: The integrative function of the basal ganglia in instrumental conditioning. *Behavioural Brain Research* 199, 43–52 (2009)
- Bar-Gad, I., Morris, G., Bergman, H.: Information processing, dimensionality reduction, and reinforcement learning in the basal ganglia. *Progress in Neurobiology* 71, 439–473 (2003)
- Barnes, T.D., Kubota, Y., Hu, D., Jin, D.Z., Graybiel, A.M.: Activity of striatal neurons reflects dynamic encoding and recoding of procedural memories. *Nature* 437, 1158–1161 (2005)

- Barto, A.G.: Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology* 4, 229–256 (1985)
- Barto, A.G.: Adaptive critics and the basal ganglia. In: Houk, J.C., Davis, J.L., Beiser, D.G. (eds.) *Models of Information Processing in the Basal Ganglia*, ch. 11, pp. 215–232. MIT Press, Cambridge (1995)
- Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13, 341–379 (2003)
- Barto, A.G., Sutton, R.S.: Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioral Brain Research* 4, 221–235 (1982)
- Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13, 835–846 (1983)
- Bayer, H.M., Glimcher, P.W.: Midbrain dopamine neurons encode a quantitative reward prediction error signal. *Neuron* 47, 129–141 (2005)
- Belin, D., Jonkman, S., Dickinson, A., Robbins, T.W., Everitt, B.J.: Parallel and interactive learning processes within the basal ganglia: relevance for the understanding of addiction. *Behavioural Brain Research* 199, 89–102 (2009)
- Berridge, K.C.: The debate over dopamine's role in reward: The case for incentive salience. *Psychopharmacology* 191, 391–431 (2007)
- Berridge, K.C., Robinson, T.E.: What is the role of dopamine in reward: Hedonic impact, reward learning, or incentive salience? *Brain Research Reviews* 28, 309–369 (1998)
- Berridge, K.C., Robinson, T.E., Aldridge, J.W.: Dissecting components of reward: 'Liking,' 'wanting,' and learning. *Current Opinion in Pharmacology* 9, 65–73 (2009)
- Björklund, A., Dunnett, S.B.: Dopamine neuron systems in the brain: an update. *Trends in Neurosciences* 30, 194–202 (2007)
- Bogacz, R., Gurney, K.: The basal ganglia and cortex implement optimal decision making between alternative actions. *Neural Computation* 19, 442–477 (2007)
- Botvinick, M.M., Niv, Y., Barto, A.G.: Hierarchically organized behavior and its neural foundations: A reinforcement-learning perspective. *Cognition* 113, 262–280 (2009)
- Brandon, S.E., Vogel, E.G., Wagner, A.R.: Computational theories of classical conditioning. In: Moore, J.W. (ed.) *A Neuroscientist's Guide to Classical Conditioning*, ch. 7, pp. 232–310. Springer, New York (2002)
- Bromberg-Martin, E.S., Matsumoto, M., Hikosaka, O.: Dopamine in motivational control: Rewarding, aversive, and alerting. *Neuron* 68, 815–834 (2010)
- Brown, P.L., Jenkins, H.M.: Auto-shaping of the pigeon's key-peck. *Journal of the Experimental Analysis of Behavior* 11, 1–8 (1968)
- Calabresi, P., Picconi, B., Tozzi, A., DiFilippo, M.: Dopamine-mediated regulation of corticostriatal synaptic plasticity. *Trends in Neuroscience* 30, 211–219 (2007)
- Cannon, C.M., Palmeriter, R.D.: Reward without dopamine. *Journal of Neuroscience* 23, 10,827–10,831 (2003)
- Cardinal, R.N., Parkinson, J.A., Hall, J., Everitt, B.J.: Emotion and motivation: The role of the amygdala, ventral striatum, and prefrontal cortex. *Neuroscience and Biobehavioural Reviews* 26, 321–352 (2002)
- Cohen, M.X.: Neurocomputational mechanisms of reinforcement-guided learning in humans: a review. *Cognitive, Affective, and Behavioral Neuroscience* 8, 113–125 (2008)
- Cohen, M.X., Frank, M.J.: Neurocomputational models of the basal ganglia in learning, memory, and choice. *Behavioural Brain Research* 199, 141–156 (2009)
- Corrado, G., Doya, K.: Understanding neural coding through the model-based analysis of decision-making. *The Journal of Neuroscience* 27, 8178–8180 (2007)

- Daw, N.D., Doya, K.: The computational neurobiology of learning and reward. *Current Opinion in Neurobiology* 16, 199–204 (2006)
- Daw, N.D., Touretzky, D.S.: Long-term reward prediction in TD models of the dopamine system. *Neural Computation* 14, 2567–2583 (2002)
- Daw, N.D., Kakade, S., Dayan, P.: Opponent interactions between serotonin and dopamine. *Neural Networks* 15, 603–616 (2002)
- Daw, N.D., Niv, Y., Dayan, P.: Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience* 8, 1704–1711 (2005)
- Daw, N.D., Courville, A.C., Touretzky, D.S.: Representation and timing in theories of the dopamine system. *Neural Computation* 18, 1637–1677 (2006a)
- Daw, N.D., O'Doherty, J.P., Dayan, P., Seymour, B., Dolan, R.J.: Cortical substrates for exploratory decisions in humans. *Nature* 441, 876–879 (2006b)
- Dayan, P., Daw, N.D.: Connections between computational and neurobiological perspectives on decision making. *Cognitive, Affective, and Behavioral Neuroscience* 8, 429–453 (2008)
- Dayan, P., Niv, Y.: Reinforcement learning: the good, the bad, and the ugly. *Current Opinion in Neurobiology* 18, 185–196 (2008)
- Dayan, P., Niv, Y., Seymour, B., Daw, N.D.: The misbehavior of value and the discipline of the will. *Neural Networks* 19, 1153–1160 (2006)
- Dickinson, A.: Actions and habits: the development of behavioural autonomy. *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 308, 67–78 (1985)
- Dickinson, A., Balleine, B.W.: Motivational control of goal-directed action. *Animal Learning and Behavior* 22, 1–18 (1994)
- Doll, B.B., Frank, M.J.: The basal ganglia in reward and decision making: computational models and empirical studies. In: Dreher, J., Tremblay, L. (eds.) *Handbook of Reward and Decision Making*, ch. 19, pp. 399–425. Academic Press, Oxford (2009)
- Dommett, E., Coizet, V., Blaha, C.D., Martindale, J., Lefebvre, V., Mayhew, N.W.J.E., Overton, P.G., Redgrave, P.: How visual stimuli activate dopaminergic neurons at short latency. *Science* 307, 1476–1479 (2005)
- Doya, K.: What are the computations of the cerebellum, the basal ganglia, and the cerebral cortex? *Neural Networks* 12, 961–974 (1999)
- Doya, K.: Reinforcement learning: Computational theory and biological mechanisms. *HFSP Journal* 1, 30–40 (2007)
- Doya, K.: Modulators of decision making. *Nature Neuroscience* 11, 410–416 (2008)
- Doyon, J., Bellec, P., Amsel, R., Penhune, V., Monchi, O., Carrier, J., Lehéricy, S., Benali, H.: Contributions of the basal ganglia and functionally related brain structures to motor learning. *Behavioural Brain Research* 199, 61–75 (2009)
- Eckerman, D.A., Hienz, R.D., Stern, S., Kowlowitz, V.: Shaping the location of a pigeon's peck: Effect of rate and size of shaping steps. *Journal of the Experimental Analysis of Behavior* 33, 299–310 (1980)
- Ferster, C.B., Skinner, B.F.: *Schedules of Reinforcement*. Appleton-Century-Crofts, New York (1957)
- Fiorillo, C.D., Tobler, P.N., Schultz, W.: Discrete coding of reward probability and uncertainty by dopamine neurons. *Science* 299, 1898–1902 (2003)
- Frank, M.J.: Dynamic dopamine modulation in the basal ganglia: a neurocomputational account of cognitive deficits in medicated and nonmedicated Parkinsonism. *Journal of Cognitive Neuroscience* 17, 51–72 (2005)

- Frank, M.J., Claus, E.D.: Anatomy of a decision: Striato-orbitofrontal interactions in reinforcement learning, decision making, and reversal. *Psychological Review* 113, 300–326 (2006)
- Frank, M.J., Seeberger, L.C., O'Reilly, R.C.: By carrot or by stick: Cognitive reinforcement learning in parkinsonism. *Science* 306, 1940–1943 (2004)
- Gardner, R.: Multiple-choice decision behavior. *American Journal of Psychology* 71, 710–717 (1958)
- Gläscher, J.P., O'Doherty, J.P.: Model-based approaches to neuroimaging combining reinforcement learning theory with fMRI data. *Wiley Interdisciplinary Reviews: Cognitive Science* 1, 501–510 (2010)
- Gläscher, J.P., Daw, N.D., Dayan, P., O'Doherty, J.P.: States versus rewards: Dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron* 66, 585–595 (2010)
- Glimcher, P.W.: Decisions, Uncertainty, and the Brain: The Science of Neuroeconomics. MIT Press, Cambridge (2003)
- Glimcher, P.W., Rustichini, A.: Neuroeconomics: The consilience of brain and decision. *Science* 306, 447–452 (2004)
- Gluck, M.A.: Behavioral and neural correlates of error correction in classical conditioning and human category learning. In: Gluck, M.A., Anderson, J.R., Kosslyn, S.M. (eds.) *Memory and Mind: A Festschrift for Gordon H. Bower*, ch. 18, pp. 281–305. Lawrence Erlbaum Associates, New York (2008)
- Gold, J.I., Shadlen, M.N.: The neural basis of decision making. *Annual Review of Neuroscience* 30, 535–574 (2007)
- Goldman-Rakic, P.S.: Cellular basis of working memory. *Neuron* 14, 447–485 (1995)
- Goodnow, J.T.: Determinants of choice-distribution in two-choice situations. *The American Journal of Psychology* 68, 106–116 (1955)
- Gomezano, I., Schneiderman, N., Deaux, E.G., Fuentes, I.: Nictitating membrane: Classical conditioning and extinction in the albino rabbit. *Science* 138, 33–34 (1962)
- Grafton, S.T., Hamilton, A.F.: Evidence for a distributed hierarchy of action representation in the brain. *Human Movement Science* 26, 590–616 (2007)
- Graybiel, A.M.: The basal ganglia: learning new tricks and loving it. *Current Opinion in Neurobiology* 15, 638–644 (2005)
- Graybiel, A.M.: Habits, rituals, and the evaluative brain. *Annual Review of Neuroscience* 31, 359–387 (2008)
- Graybiel, A.M., Aosaki, T., Flaherty, A.W., Kimura, M.: The basal ganglia and adaptive motor control. *Science* 265, 1826–1831 (1994)
- Green, L., Myerson, J.: A discounting framework for choice with delayed and probabilistic rewards. *Psychological Bulletin* 130, 769–792 (2004)
- Grupen, R., Huber, M.: A framework for the development of robot behavior. In: 2005 AAAI Spring Symposium Series: Developmental Robotics. American Association for the Advancement of Artificial Intelligence, Palo Alto (2005)
- Gurney, K.: Reverse engineering the vertebrate brain: Methodological principles for a biologically grounded programme of cognitive modelling. *Cognitive Computation* 1, 29–41 (2009)
- Gurney, K., Prescott, T.J., Redgrave, P.: A computational model of action selection in the basal ganglia. I. A new functional anatomy. *Biological Cybernetics* 84, 401–410 (2001)
- Gurney, K., Prescott, T.J., Wickens, J.R., Redgrave, P.: Computational models of the basal ganglia: From robots to membranes. *Trends in Neuroscience* 27, 453–459 (2004)

- Haber, S.N.: The primate basal ganglia: Parallel and integrative networks. *Journal of Chemical Neuroanatomy* 26, 317–330 (2003)
- Haber, S.N., Kim, K.S., Mailly, P., Calzavara, R.: Reward-related cortical inputs define a large striatal region in primates that interface with associative cortical inputs, providing a substrate for incentive-based learning. *The Journal of Neuroscience* 26, 8368–8376 (2006)
- Haruno, M., Kawato, M.: Heterarchical reinforcement-learning model for integration of multiple cortico-striatal loops: fMRI examination in stimulus-action-reward association learning. *Neural Networks* 19, 1242–1254 (2006)
- Hazy, T.E., Frank, M.J., O'Reilly, R.C.: Neural mechanisms of acquired phasic dopamine responses in learning. *Neuroscience and Biobehavioral Reviews* 34, 701–720 (2010)
- Herrnstein, R.J.: Relative and absolute strength of response as a function of frequency of reinforcement. *Journal of the Experimental Analysis of Behavior* 4, 267–272 (1961)
- Hikosaka, O.: Basal ganglia mechanisms of reward-oriented eye movement. *Annals of the New York Academy of Science* 1104, 229–249 (2007)
- Hollerman, J.R., Schultz, W.: Dopamine neurons report an error in the temporal prediction of reward during learning. *Nature Neuroscience* 1, 304–309 (1998)
- Horvitz, J.C.: Mesolimbocortical and nigrostriatal dopamine responses to salient non-reward events. *Neuroscience* 96, 651–656 (2000)
- Houk, J.C., Wise, S.P.: Distributed modular architectures linking basal ganglia, cerebellum, and cerebral cortex: Their role in planning and controlling action. *Cerebral Cortex* 5, 95–110 (1995)
- Houk, J.C., Adams, J.L., Barto, A.G.: A model of how the basal ganglia generate and use neural signals that predict reinforcement. In: Houk, J.C., Davis, J.L., Beiser, D.G. (eds.) *Models of Information Processing in the Basal Ganglia*, ch. 13, pp. 249–270. MIT Press, Cambridge (1995)
- Houk, J.C., Bastianen, C., Fansler, D., Fishbach, A., Fraser, D., Reber, P.J., Roy, S.A., Simo, L.S.: Action selection and refinement in subcortical loops through basal ganglia and cerebellum. *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 362, 1573–1583 (2007)
- Hull, C.L.: *Principles of Behavior*. Appleton-Century-Crofts, New York (1943)
- Humphries, M.D., Prescott, T.J.: The ventral basal ganglia, a selection mechanism at the crossroads of space, strategy, and reward. *Progress in Neurobiology* 90, 385–417 (2010)
- Ito, M., Doya, K.: Validation of decision-making models and analysis of decision variables in the rat basal ganglia. *The Journal of Neuroscience* 29, 9861–9874 (2009)
- Joel, D., Weiner, I.: The organization of the basal ganglia-thalamocortical circuits: Open interconnected rather than closed segregated. *Neuroscience* 63, 363–379 (1994)
- Joel, D., Niv, Y., Ruppin, E.: Actor-critic models of the basal ganglia: New anatomical and computational perspectives. *Neural Networks* 15, 535–547 (2002)
- Joshua, M., Adler, A., Bergman, H.: The dynamics of dopamine in control of motor behavior. *Current Opinion in Neurobiology* 19, 615–620 (2009)
- Kamin, L.J.: Predictability, surprise, attention, and conditioning. In: Campbell, B.A., Church, R.M. (eds.) *Punishment and Aversive Behavior*, pp. 279–296. Appleton-Century-Crofts, New York (1969)
- Kehoe, E.J., Schreurs, B.G., Graham, P.: Temporal primacy overrides prior training in serial compound conditioning of the rabbit's nictitating membrane response. *Animal Learning and Behavior* 15, 455–464 (1987)
- Kim, H., Sul, J.H., Huh, N., Lee, D., Jung, M.W.: Role of striatum in updating values of chosen actions. *The Journal of Neuroscience* 29, 14,701–14,712 (2009)

- Kishida, K.T., King-Casas, B., Montague, P.R.: Neuroeconomic approaches to mental disorders. *Neuron* 67, 543–554 (2010)
- Klopff, A.H.: *The Hedonistic Neuron: A Theory of Memory, Learning and Intelligence*. Hemisphere Publishing Corporation, Washington DC (1982)
- Kobayashi, S., Schultz, W.: Influence of reward delays on responses of dopamine neurons. *The Journal of Neuroscience* 28, 7837–7846 (2008)
- Konidaris, G.D., Barto, A.G.: Skill discovery in continuous reinforcement learning domains using skill chaining. In: Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C.K.I., Culotta, A. (eds.) *Advances in Neural Information Processing Systems (NIPS)*, vol. 22, pp. 1015–1023. MIT Press, Cambridge (2009)
- Lau, B., Glimcher, P.W.: Value representations in the primate striatum during matching behavior. *Neuron* 58, 451–463 (2008)
- Ljungberg, T., Apicella, P., Schultz, W.: Responses of monkey dopamine neurons during learning of behavioral reactions. *Journal of Neurophysiology* 67, 145–163 (1992)
- Ludvig, E.A., Sutton, R.S., Kehoe, E.J.: Stimulus representation and the timing of reward-prediction errors in models of the dopamine system. *Neural Computation* 20, 3034–3054 (2008)
- Maia, T.V.: Reinforcement learning, conditioning, and the brain: Successes and challenges. *Cognitive, Affective, and Behavioral Neuroscience* 9, 343–364 (2009)
- Maia, T.V., Frank, M.J.: From reinforcement learning models to psychiatric and neurobiological disorders. *Nature Neuroscience* 14, 154–162 (2011)
- Matsumoto, K., Suzuki, W., Tanaka, K.: Neuronal correlates of goal-based motor selection in the prefrontal cortex. *Science* 301, 229–232 (2003)
- Matsuzaka, Y., Picard, N., Strick, P.: Skill representation in the primary motor cortex after long-term practice. *Journal of Neurophysiology* 97, 1819–1832 (2007)
- McHaffie, J.G., Stanford, T.R., Stein, B.E., Coizet, V., Redgrave, P.: Subcortical loops through the basal ganglia. *Trends in Neurosciences* 28, 401–407 (2005)
- Middleton, F.A., Strick, P.L.: Basal-ganglia “projections” to the prefrontal cortex of the primate. *Cerebral Cortex* 12, 926–935 (2002)
- Miller, E.K., Cohen, J.D.: An integrative theory of prefrontal cortex function. *Annual Review of Neuroscience* 24, 167–202 (2001)
- Miller, J.D., Sanghera, M.K., German, D.C.: Mesencephalic dopaminergic unit activity in the behaviorally conditioned rat. *Life Sciences* 29, 1255–1263 (1981)
- Mink, J.W.: The basal ganglia: Focused selection and inhibition of competing motor programs. *Progress in Neurobiology* 50, 381–425 (1996)
- Mirolli, M., Mannella, F., Baldassarre, G.: The roles of the amygdala in the affective regulation of body, brain, and behaviour. *Connection Science* 22, 215–245 (2010)
- Montague, P.R., Dayan, P., Sejnowski, T.J.: A framework for mesencephalic dopamine systems based on predictive Hebbian learning. *Journal of Neuroscience* 16, 1936–1947 (1996)
- Montague, P.R., Hyman, S.E., Cohen, J.D.: Computational roles for dopamine in behavioural control. *Nature* 431, 760–767 (2004)
- Montague, P.R., King-Casas, B., Cohen, J.D.: Imaging valuation models in human choice. *Annual Review of Neuroscience* 29, 417–448 (2006)
- Moore, J.W., Choi, J.S.: Conditioned response timing and integration in the cerebellum. *Learning and Memory* 4, 116–129 (1997)
- Morris, G., Nevet, A., Arkadir, D., Vaadia, E., Bergman, H.: Midbrain dopamine neurons encode decisions for future action. *Nature Neuroscience* 9, 1057–1063 (2006)

- Mushiake, H., Saito, N., Sakamoto, K., Itoyama, Y., Tanji, J.: Activity in the lateral prefrontal cortex reflects multiple steps of future events in action plans. *Neuron* 50, 631–641 (2006)
- Nakahara, H., Itoh, H., Kawagoe, R., Takikawa, Y., Hikosaka, O.: Dopamine neurons can represent context-dependent prediction error. *Neuron* 41, 269–280 (2004)
- Ng, A., Harada, D., Russell, S.: Policy invariance under reward transformations: theory and applications to reward shaping. In: *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287 (1999)
- Nicola, S.M.: The nucleus accumbens as part of a basal ganglia action selection circuit. *Psychopharmacology* 191, 521–550 (2007)
- Niv, Y.: Reinforcement learning in the brain. *Journal of Mathematical Psychology* 53, 139–154 (2009)
- Niv, Y., Duff, M.O., Dayan, P.: Dopamine, uncertainty, and TD learning. *Behavioral and Brain Functions* 1, 6 (2005)
- Niv, Y., Daw, N.D., Dayan, P.: Choice values. *Nature Neuroscience* 9, 987–988 (2006a)
- Niv, Y., Joel, D., Dayan, P.: A normative perspective on motivation. *Trends in Cognitive Sciences* 10, 375–381 (2006b)
- Nomoto, K., Schultz, W., Watanabe, T., Sakagami, M.: Temporally extended dopamine responses to perceptually demanding reward-predictive stimuli. *The Journal of Neuroscience* 30, 10,692–10,702 (2010)
- O'Doherty, J.P., Dayan, P., Schultz, J., Deichmann, R., Friston, K., Dolan, R.J.: Dissociable roles of ventral and dorsal striatum in instrumental conditioning. *Science* 304, 452–454 (2004)
- Olds, J., Milner, P.: Positive reinforcement produced by electrical stimulation of septal area and other regions of rat brain. *Journal of Comparative and Physiological Psychology* 47, 419–427 (1954)
- O'Reilly, R.C., Frank, M.J.: Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia. *Neural Computation* 18, 283–328 (2006)
- Packard, M.G., Knowlton, B.J.: Learning and memory functions of the basal ganglia. *Annual Review of Neuroscience* 25, 563–593 (2002)
- Pasupathy, A., Miller, E.K.: Different time courses of learning-related activity in the prefrontal cortex and striatum. *Nature* 433, 873–876 (2005)
- Pavlov, I.P.: *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex*. Oxford University Press, Toronto (1927)
- Pennartz, C.M., Berke, J.D., Graybiel, A.M., Ito, R., Lansink, C.S., van der Meer, M., Redish, A.D., Smith, K.S., Voorn, P.: Corticostriatal interactions during learning, memory processing, and decision making. *The Journal of Neuroscience* 29, 12,831–12,838 (2009)
- Pessiglione, M., Seymour, B., Flandin, G., Dolan, R.J., Frith, C.D.: Dopamine-dependent prediction errors underpin reward-seeking behaviour in humans. *Nature* 442, 1042–1045 (2006)
- Phelps, E.A., LeDoux, J.E.: Contributions of the amygdala to emotion processing: From animal models to human behavior. *Neuron* 48, 175–187 (2005)
- Poldrack, R.A., Sabb, F.W., Foerde, K., Tom, S.M., Asarnow, R.F., Bookheimer, S.Y., Knowlton, B.J.: The neural correlates of motor skill automaticity. *The Journal of Neuroscience* 25, 5356–5364 (2005)
- Pompilio, L., Kacelnik, A.: State-dependent learning and suboptimal choice: when starlings prefer long over short delays to food. *Animal Behaviour* 70, 571–578 (2005)
- Redgrave, P., Gurney, K.: The short-latency dopamine signal: a role in discovering novel actions? *Nature Reviews Neuroscience* 7, 967–975 (2006)

- Redgrave, P., Gurney, K., Reynolds, J.: What is reinforced by phasic dopamine signals? *Brain Research Reviews* 58, 322–339 (2008)
- Redgrave, P., Rodriguez, M., Smith, Y., Rodriguez-Oroz, M.C., Lehericy, S., Bergman, H., Agid, Y., DeLong, M.R., Obeso, J.A.: Goal-directed and habitual control in the basal ganglia: implications for Parkinson's disease. *Nature Reviews Neuroscience* 11, 760–772 (2010)
- Redish, A.D., Jensen, S., Johnson, A.: A unified framework for addiction: Vulnerabilities in the decision process. *Behavioral and Brain Sciences* 31, 415–487 (2008)
- Rescorla, R.A., Wagner, A.R.: A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In: Black, A.H., Prokasy, W.F. (eds.) *Classical Conditioning II: Current Research and Theory*, pp. 64–99. Appleton-Century-Crofts, New York (1972)
- Richardson, W.K., Warzak, W.J.: Stimulus stringing by pigeons. *Journal of the Experimental Analysis of Behavior* 36, 267–276 (1981)
- Roesch, M.R., Calu, D.J., Schoenbaum, G.: Dopamine neurons encode the better option in rats deciding between differently delayed or sized rewards. *Nature Neuroscience* 10, 1615–1624 (2007)
- Roesch, M.R., Singh, T., Brown, P.L., Mullins, S.E., Schoenbaum, G.: Ventral striatal neurons encode the value of the chosen action in rats deciding between differently delayed or sized rewards. *The Journal of Neuroscience* 29, 13,365–13,376 (2009)
- Samejima, K., Doya, K.: Multiple representations of belief states and action values in corticobasal ganglia loops. *Annals of the New York Academy of Sciences* 1104, 213–228 (2007)
- Samejima, K., Ueda, Y., Doya, K., Kimura, M.: Representation of action-specific reward values in the striatum. *Science* 310, 1337–1340 (2005)
- Satoh, T., Nakai, S., Sato, T., Kimura, M.: Correlated coding of motivation and outcome of decision by dopamine neurons. *The Journal of Neuroscience* 23, 9913–9923 (2003)
- Schultz, W.: Responses of midbrain dopamine neurons to behavioral trigger stimuli in the monkey. *Journal of Neurophysiology* 56, 1439–1461 (1986)
- Schultz, W.: Predictive reward signal of dopamine neurons. *Journal of Neurophysiology* 80, 1–27 (1998)
- Schultz, W.: Behavioral theories and the neurophysiology of reward. *Annual Review of Psychology* 57, 8–115 (2006)
- Schultz, W.: Multiple dopamine functions at different time courses. *Annual Review of Neuroscience* 30, 259–288 (2007)
- Schultz, W.: Dopamine signals for reward value and risk: basic and recent data. *Behavioral and Brain Functions* 6, 24 (2010)
- Schultz, W., Apicella, P., Ljungberg, T.: Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task. *The Journal of Neuroscience* 13, 900–913 (1993)
- Schultz, W., Dayan, P., Montague, P.R.: A neural substrate of prediction and reward. *Science* 275, 1593–1599 (1997)
- Schultz, W., Tremblay, L., Hollerman, J.R.: Changes in behavior-related neuronal activity in the striatum during learning. *Trends in Neuroscience* 26, 321–328 (2003)
- Seger, C.A., Miller, E.K.: Category learning in the brain. *Annual Review of Neuroscience* 33, 203–219 (2010)
- Selfridge, O.J., Sutton, R.S., Barto, A.G.: Training and tracking in robotics. In: Joshi, A. (ed.) *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 670–672. Morgan Kaufmann, San Mateo (1985)

- Shah, A.: Biologically-based functional mechanisms of motor skill acquisition. PhD thesis, University of Massachusetts Amherst (2008)
- Shah, A., Barto, A.G.: Effect on movement selection of an evolving sensory representation: A multiple controller model of skill acquisition. *Brain Research* 1299, 55–73 (2009)
- Shanks, D.R., Tunney, R.J., McCarthy, J.D.: A re-examination of probability matching and rational choice. *Journal of Behavioral Decision Making* 15, 233–250 (2002)
- Siegel, S., Goldstein, D.A.: Decision making behaviour in a two-choice uncertain outcome situation. *Journal of Experimental Psychology* 57, 37–42 (1959)
- Skinner, B.F.: *The Behavior of Organisms*. Appleton-Century-Crofts, New York (1938)
- Staddon, J.E.R., Cerutti, D.T.: Operant behavior. *Annual Review of Psychology* 54, 115–144 (2003)
- Sutton, R.S.: Learning to predict by methods of temporal differences. *Machine Learning* 3, 9–44 (1988)
- Sutton, R.S., Barto, A.G.: Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review* 88, 135–170 (1981)
- Sutton, R.S., Barto, A.G.: A temporal-difference model of classical conditioning. In: *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 355–378 (1987)
- Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge (1998)
- Tanji, J., Hoshi, E.: Role of the lateral prefrontal cortex in executive behavioral control. *Physiological Reviews* 88, 37–57 (2008)
- Thorndike, E.L.: *Animal Intelligence: Experimental Studies*. Macmillan, New York (1911)
- Tindell, A.J., Berridge, K.C., Zhang, J., Pecina, S., Aldridge, J.W.: Ventral pallidal neurons code incentive motivation: Amplification by mesolimbic sensitization and amphetamine. *European Journal of Neuroscience* 22, 2617–2634 (2005)
- Tobler, P.N., Dickinson, A., Schultz, W.: Coding of predicted reward omission by dopamine neurons in a conditioned inhibition paradigm. *The Journal of Neuroscience* 23, 10,402–10,410 (2003)
- Tobler, P.N., Fiorillo, C.D., Schultz, W.: Adaptive coding of reward value by dopamine neurons. *Science* 307, 1642–1645 (2005)
- Tolman, E.C.: Cognitive maps in rats and men. *The Psychological Review* 55, 189–208 (1948)
- Tolman, E.C.: There is more than one kind of learning. *Psychological Review* 56, 44–55 (1949)
- Waelti, P., Dickinson, A., Schultz, W.: Dopamine responses comply with basic assumptions of formal learning theory. *Nature* 412, 43–48 (2001)
- Wallis, J.D.: Orbitofrontal cortex and its contribution to decision-making. *Annual Review of Neuroscience* 30, 31–56 (2007)
- Watson, J.B.: *Behavior: An Introduction to Comparative Psychology*. Holt, New York (1914)
- Wickens, J.R.: Synaptic plasticity in the basal ganglia. *Behavioural Brain Research* 199, 119–128 (2009)
- Wickens, J.R., Budd, C.S., Hyland, B.I., Arbuthnott, G.W.: Striatal contributions to reward and decision making. Making sense of regional variations in a reiterated processing matrix. *Annals of the New York Academy of Sciences* 1104, 192–212 (2007)
- Widrow, B., Hoff, M.E.: Adaptive switching circuits. In: *1960 WESCON Convention Record Part IV*, pp. 96–104. Institute of Radio Engineers, New York (1960)
- Wilson, C.J.: Basal ganglia. In: Shepherd, G.M. (ed.) *The Synaptic Organization of the Brain*, ch. 9, 5th edn., pp. 361–414. Oxford University Press, Oxford (2004)

- Wise, R.A.: Dopamine, learning and motivation. *Nature Reviews Neuroscience* 5, 483–494 (2004)
- Wolpert, D.: Probabilistic models in human sensorimotor control. *Human Movement Science* 27, 511–524 (2007)
- Wörgötter, F., Porr, B.: Temporal sequence learning, prediction, and control: A review of different models and their relation to biological mechanisms. *Neural Computation* 17, 245–319 (2005)
- Wräse, J., Kahnt, T., Schlagenhauf, F., Beck, A., Cohen, M.X., Knutson, B., Heinz, A.: Different neural systems adjust motor behavior in response to reward and punishment. *NeuroImage* 36, 1253–1262 (2007)
- Wyvill, C.L., Berridge, K.C.: Intra-accumbens amphetamine increases the conditioned incentive salience of sucrose reward: Enhancement of reward “wanting” without enhanced “liking” or response reinforcement. *Journal of Neuroscience* 20, 8122–8130 (2000)
- Yin, H.H., Ostlund, S.B., Balleine, B.W.: Reward-guided learning beyond dopamine in the nucleus accumbens: the integrative functions of cortico-basal ganglia networks. *European Journal of Neuroscience* 28, 1437–1448 (2008)
- Yu, A., Dayan, P.: Uncertainty, neuromodulation and attention. *Neuron* 46, 681–692 (2005)

Chapter 17

Reinforcement Learning in Games

István Szita

Abstract. Reinforcement learning and games have a long and mutually beneficial common history. From one side, games are rich and challenging domains for testing reinforcement learning algorithms. From the other side, in several games the best computer players use reinforcement learning. The chapter begins with a selection of games and notable reinforcement learning implementations. Without any modifications, the basic reinforcement learning algorithms are rarely sufficient for high-level gameplay, so it is essential to discuss the additional ideas, ways of inserting domain knowledge, implementation decisions that are necessary for scaling up. These are reviewed in sufficient detail to understand their potentials and their limitations. The second part of the chapter lists challenges for reinforcement learning in games, together with a review of proposed solution methods. While this listing has a game-centric viewpoint, and some of the items are specific to games (like opponent modelling), a large portion of this overview can provide insight for other kinds of applications, too. In the third part we review how reinforcement learning can be useful in game development and find its way into commercial computer games. Finally, we provide pointers for more in-depth reviews of specific games and solution approaches.

17.1 Introduction

Reinforcement learning (RL) and games have a long and fruitful common history. Samuel’s *Checkers* player, one of the first learning programs ever, already had the principles of temporal difference learning, decades before temporal difference learning (TD) was described and analyzed. And it was another game, *Backgammon*, where reinforcement learning reached its first big success, when Tesauro’s

István Szita
University of Alberta, Canada
e-mail: szityu@gmail.com

TD-Gammon reached and exceeded the level of top human players – and did so entirely by learning on its own. Since then, RL has been applied to many other games, and while it could not repeat the success of TD-Gammon in every game, there are many promising results and many lessons to be learned. We hope to present these in this chapter, both in classical games and computer games games, real-time strategy games, first person shooters, role-playing games. Most notably, reinforcement learning approaches seem to have the upper hand in one of the flagship applications of artificial intelligence research, *Go*.

From a different point of view, games are an excellent testbed for RL research. Games are designed to entertain, amuse and challenge humans, so, by studying games, we can (hopefully) learn about human intelligence, and the challenges that human intelligence needs to solve. At the same time, games are challenging domains for RL algorithms as well, probably for the same reason they are so for humans: they are designed to involve interesting decisions. The types of challenges move on a wide scale, and we aim at presenting a representative selection of these challenges, together with RL approaches to tackle them.

17.1.1 Aims and Structure

One goal of this chapter is to collect notable RL applications to games. But there is another, far more important goal: to get an idea how (and why) RL algorithms work (or fail) in practice. Most of the algorithms mentioned in the chapter are described in detail in other parts of the book. Their theoretical analysis (if exists) give us guarantees that they work under ideal conditions, but these are impractical for most games: conditions are too restrictive, statements are loose or both. For example, we know that TD-learning¹ converges to an optimal policy if the environment is a finite Markov decision process (MDP), values of each state are stored individually, learning rates are decreasing in an proper manner, and exploration is sufficient. Most of these conditions are violated in a typical game application. Yet, TD-learning works phenomenally well for *Backgammon* and not at all for other games (for example, *Tetris*). There is a rich literature of attempts to identify game attributes that make TD-learning and other RL algorithms perform well. We think that an overview of these attempts is pretty helpful for future developments.

In any application of RL, the choice of algorithm is just one among many factors that determine success or failure. Oftentimes the choice of algorithm is not even the most significant factor: the choice of representation, formalization, the encoding of domain knowledge, additional heuristics and variations, proper setting of parameters can all have great influence. For each of these issues, we can find exciting ideas that have been developed for conquering specific games. Sadly (but not surprisingly), there is no “magic bullet”: all approaches are more-or-less game- or genre-specific.

¹ In theoretical works, TD-learning refers to a policy evaluation method, while in the games-related literature, it is used in the sense “actor-critic learning with a critic using TD-learning”.

Still, we believe that studying these ideas can give us useful insights, and some of the findings can be generalized to other applications.

The vast variety of games makes the topic diverse and hard to organize. To keep coherence, we begin with in-depth overview of reinforcement learning in a few well-studied games: *Backgammon*, *Chess*, *Go*, *Poker*, *Tetris* and *real-time strategy games*. These games can be considered as case studies, introducing many different issues of RL. Section 17.3 summarizes some of the notable issues and challenges. Section 17.4 is probably the part that deserves the most attention, as it surveys the practical uses of RL in games. Finally, Section 17.5 contains pointers for further reading and our conclusions.

17.1.2 Scope

In a chapter about RL applications in games, one has to draw boundaries along two dimensions: what kind of algorithms fit in? And what kind of games fit in? Neither decision is an easy one.

It is clear that TD-Gammon has a place here (and Deep Blue probably does not). But, for example, should the application of UCT to *Go* be here? We could treat it under the umbrella of planning, or even game theory, though UCT certainly has strong connections to RL. Other applications have a fuzzy status (another good example is the use of evolutionary methods to RL tasks). Following the philosophy of the book, we consider “RL applications” in a broad sense, including all approaches that take inspiration from elements of reinforcement learning. However, we had to leave out some important games, for example *Poker*, *Bridge* or the PSPACE-hard *Sokoban*, or the theory of general-sum games, which inspired lots of results in AI research, but most of that is not considered RL in general.

As for games, we do not wish to exclude any actual game that people play, so computer games, modern board games have definitely place here. Nevertheless, the chapter will be inescapably biased towards “classical” games, simply because of a larger body of research results available.

17.2 A Showcase of Games

In this section we overview the RL-related results for several games. We focus on the aspects that are relevant from the point of view of RL. So, for example, we explain game rules only to the level of detail that is necessary for understanding the particular RL issues.

17.2.1 Backgammon

In *Backgammon*, two players compete to remove their stones from the board as quickly as possible. The board has 24 fields organized linearly, and each player starts with 15 stones, arranged symmetrically. Figure 17.1 shows the board with the starting position. Players move their stones in opposite directions. If a stone reaches the end of the track, it can be removed, but only if all the player's stones are already in the last quarter. Players take turns, and they roll two dice each turn. The amount of movement is determined by the dice, but the players decide which stones to move. Single stones can be hit (and have to start the track from the beginning), but the player cannot move to a field with two or more opponent stones.²



Fig. 17.1 Backgammon board with the initial game setup, two sets of dice and the doubling cube

Basic strategies include blocking of key positions, building long blocks that are hard or impossible to jump through, and racing to the end. On a higher level, players need to estimate the probabilities of various events fairly accurately (among other things). Another element of strategy is added by the “doubling cube”: any player can offer to double the stakes of the game. The other player has two options: he may accept the doubling, and get the cube and the right to offer the next doubling, or he may give up the game, losing the current stakes only. Knowing when to offer doubling and when to accept is the hardest part of *Backgammon*.

Traditional search algorithms are not effective for *Backgammon* because of its chance element: for each turn, there are 21 possible outcomes for the dice rolls, with 20 legal moves on average for each of them. This gives a branching factor over 400, making deep lookahead search impossible.

² For a detailed description of rules, please refer to one of the many backgammon resources on the internet, for example, <http://www.play65.com/Backgammon.html>

17.2.1.1 TD-Gammon

The first version of TD-Gammon (Tesauro, 1992, 2002) used a neural network as a position evaluator. The input of the network was an encoding of the game position, and its output was the value of the position.³ After rolling the dice, the program tried all legal moves and evaluated the resulting positions, then took the highest rated move. The network did not consider the possibility of doubling, that was handled by a separate heuristic formula. Rewards were +1 for winning and 0 for losing,⁴ so output values approximated the probability of winning. For each player, each board position was represented by four neurons, using a truncated unary encoding (input k was 1 if the field had at least k stones, and 0 otherwise). Later versions of TD-Gammon also used inputs that encoded expert features, e.g., “pip count”, a heuristic progress measure (the total distance of pieces from the goal).

After each step, the output of the neuron was updated according to the TD rule, and the weights were updated with the backpropagation rule. The network was trained by self-play and no explicit exploration.

TD-Gammon did surprisingly well. Tesauro’s previous backgammon program, Neurogammon, used neural networks with the same architecture, but was trained on samples labeled by human backgammon experts. TD-Gammon became significantly stronger than Neurogammon, and it was on par with Neurogammon even when handicapped, receiving only the raw table representation as input, without the features encoding domain-knowledge. In later versions, Tesauro increased the number of hidden neurons, the number of training games (up to 1500000), added 3-ply lookahead, and improved the representation. With these changes, TD-Gammon 3.0 became a world-class player.

Since then, TD-Gammon has retired, but $\text{TD}(\lambda)$ is still the basis of today’s strongest Backgammon-playing programs, Snowie, Jellyfish, Bgblitz and GNUbg. Little is known about the details because all of these programs except GNUbg are closed-source commercial products (and the techniques of GNUbg are not well-documented either). There seems to be a consensus that Snowie is the strongest of these, but no reliable rankings exist. Comparing to humans is not easy because of the high variance of the outcomes (an indicative example: in the 100-game tournament between TD-Gammon and Malcolm Davis in 1998, TD-Gammon won 99 games, yet still lost the tournament because it lost so many points in its single lost game). As an alternative to many-game tournaments, rollout analysis is used widely to compare players. A computer Backgammon program tries to find the best move for each step of the game by simulating thousands of games from the current state, and calculates how much worse the player’s actual choice was. According to

³ Actually, there were four output neurons, one for each combination of the players having/not having the situation called “gammon”, when one player wins before the other starts to remove his own stones. For the sake of simplicity, we consider only the case when neither player has a gammon.

⁴ Again, the situation considered by Tesauro was slightly more complicated: certain victories in *Backgammon* are worth +2 points, or even +3 in rare cases; furthermore, the point values can be doubled several times.

rollout analysis, Backgammon programs have exceeded the playing strength of the best human players (Tesauro, 2002).

17.2.1.2 Why Did TD-Gammon Work?

The success of TD-Gammon definitely had a great role in making $\text{TD}(\lambda)$ popular. TD-learning has found many applications, with many games among them (see Ghory, 2004, for an overview). In many of these games, TD could not repeat the same level of success, which makes the performance of TD-Gammon even more surprising. TD-learning in *Backgammon* has to deal with the RL problem in its full generality: there are stochastic transitions, stochastic and delayed rewards and a huge state space. Because of self-play, the environment is not really an MDP (the opponent changes continuously). Even more troubling is the use of nonlinear function approximation Furthermore, TD-Gammon was not doing any exploration, it always chose the greedy action, potentially resulting in poor performance. Many authors have tried to identify the reasons why TD works so well in *Backgammon*, including Tesauro (1995, 1998), Pollack and Blair (1997), Ghory (2004) and Wiering (2010). We try to summarize their arguments here.

Representation. As emphasized in the opening chapter of the book, the proper choice of representation is crucial to any RL application. Tesauro claimed that the state representation of TD-Gammon is “knowledge-free”, as it could have been formed without any knowledge of the game. On the other hand, the truncated unary representation seems to fit *Backgammon* well, just like the natural board representation (one unit per each field and player). And of course, the addition of more informative features increased playing strength considerably. The suitability of representation is supported by the fact that Neurogammon, which used the same features, but no RL, was also a reasonably good computer player.

Randomness. Every source agrees that randomness in the game makes the learning task easier. Firstly, it smoothens the value function: the value of “similar” states is close to each other, because random fluctuations can mask the differences. Function approximation works better for smooth target functions. Secondly, randomness helps solve the exploration-exploitation dilemma, or in this case, to evade it completely (in TD-Gammon, both learning agents choose the greedy action all the time). Because of the effect of dice, the players will visit large parts of the state space without any extra effort spent on exploration. As a third factor, human players are reportedly not very good in estimating probabilities, which makes them easier to defeat.

Training regime. TD-Gammon was trained by playing against itself. In general, this can be both good and bad: the agent plays against an opponent of the same level, but, on the other hand, learning may get stuck and the agent may specialize to a strategy against a weak player (itself), as it never gets to see a strong opponent. The pros and cons of self-play will be analyzed in detail in section 17.3.3.1, but we note here that the “automatic exploration” property of *Backgammon* helps prevent

learning from getting stuck: even if the opponent is weak, it may arrive at a strong position by luck. This creates a driving force for learning to get even.

Randomness and “automatic exploration” makes *Backgammon* quite unique among games. Pollack and Blair (1997) actually argued that these special properties of *Backgammon* contribute more to the success than TD-learning and neural networks. They train a linear architecture with simple gradient descent which reaches reasonably good performance. Tesauro (1998) claims, on the other hand, that this is insufficient, as neural networks have actually discovered important features that linear architectures cannot capture.

Parameters. The parameter λ of $\text{TD}(\lambda)$ determines the amount of bootstrapping. It is general consensus that intermediate values of λ work better than either extremes 0 (full bootstrapping) or 1 (no bootstrapping). Tesauro used $\lambda = 0.7$ for the initial experiments, but later switched to $\lambda = 0$ because he noticed no significant difference in performance, and with no eligibility traces, the TD update rule gets simpler and faster to compute. On the other hand, Wiering (2010) reported that for the initial learning progress is fastest with a high λ (≈ 0.8), while lower values (but still ≈ 0.6) are optimal on the long run.

Proper architecture. Tesauro (2002) describes other parts of the architecture. We emphasize one thing here: temporal difference learning is an important factor in the success of TD-Gammon, but just as important was to know *how* to use it. The neural network is trained to predict the probability of victory from a given state – and its predictions are not very accurate, it can be biased by as much as 0.1, way too much for direct use. Luckily, the bias does not change much among similar game states, so the neural network predictor can still be used to *compare and rank* the values of legal moves from a given position, as the relative values are reliable most of the time. However, we need to know the exact victory probabilities to handle the doubling cube, that is, to decide when to offer the doubling of stakes and when to accept it. For this reason, TD-Gammon used an independent heuristic formula to make doubling decisions. In some cases it is possible to calculate the probabilities of certain events exactly (e.g., in endgame situations or getting through a block). Later versions of TD-Gammon used pre-calculated probabilities as input features, and today’s leading Backgammon programs probably use both precalculated probabilities and endgame databases.

17.2.2 *Chess*

Chess is an archetypical two-player, full-information, zero-sum game.⁵ Its popularity, prestige and complexity made it a premier target of AI research for decades. In their famous 1997 tournament, *Deep Blue* defeated Gary Kasparov, world champion at the time (Hsu, 2002). Since then, chess programs have far exceeded the best humans: the current leader of FIDE rankings, Magnus Carlsen, is rated 2826 Elo,

⁵ For detailed rules, see e.g., http://en.wikipedia.org/wiki/Rules_of_chess#Play_of_the_game

while the winner of the Computer Chess Olympiad, *Rybka*, is estimated to be well above 3000.⁶

Chess is one of the top-tier games where RL has not been very successful so far, and not for the lack of trying. Initial attempts to reinforcement-learn *Chess* include Gherrity (1993) who applied Q-learning with very limited success. We review two programs here, NeuroChess of Thrun (1995) and TD-Leaf of Baxter et al (2000).



Fig. 17.2 Chessboard with a Sicilian opening. (source: Wikimedia Commons, author: Rmrfstar)

17.2.2.1 NeuroChess

NeuroChess by Thrun (1995) trained two neural networks, one representing the value function V and another one representing a predictive model M . For any game position s , $M(s)$ was an approximation of the *expected* encoding of the game state two half-steps later. Each chess board was encoded as a 175-element vector of hand-coded features (probably containing standard features of chess evaluation functions, e.g., piece values, piece-square values, penalties for double pawn, etc.) The evaluation function network was trained by an interesting extension of the temporal difference method: not only the value function was adjusted toward the target value, but also its *slope* was adjusted towards the slope of the target value function. This is supposed to give better fit. Let s_1, s_2, \dots be the sequence of states where it is white's turn (black's turns can be handled similarly), and consider a time step t . If t is the final step, then the target value for $V(s_t)$ is the final outcome of the game (0 or ± 1). In concordance with the TD(0) update rule, for non-final t , the target value is $V^{\text{target}} = \gamma V(x_{t+1})$ (note that all immediate rewards are 0). The target slope is given by

$$dV^{\text{target}} = \frac{\partial V^{\text{target}}}{\partial x_t} = \gamma \frac{\partial V(x_{t+1})}{\partial x_t} = \gamma \frac{\partial V(x_{t+1})}{\partial x_{t+1}} \frac{\partial x_{t+1}}{\partial x_t} \approx \gamma \frac{\partial V(x_{t+1})}{\partial x_{t+1}} \frac{\partial M(x_t)}{\partial x_t},$$

where the slopes of V and M can be easily extracted from the neural networks. The predictive model was trained by (supervised) backpropagation on 120000 grandmaster games, while V was trained on the temporal differences using both the target values and their slopes, using an algorithm called Tangent-Prop. The TD training lasted for 2400 games. Other details of the architecture are unknown, but it is worth mentioning a number of tricks for improving performance. According to Thrun, two

⁶ Rybka is rated above 3200 on CCRL's ranking of computer chess players

http://computerchess.org.uk/ccrl/4040/rating_list_all.html, but Elo rankings with different player pools cannot be compared directly.

of these tricks were most influential: firstly, each game starts with a random number of steps from a game database, then finished by self-play. The initialization ensures that learning effort is concentrated on interesting game situations. Secondly, the features were designed to give a much smoother representation of the game board than a raw representation (that is, the feature vectors of similar game positions were typically close to each other). Further tricks included quiescence search,⁷ discounting (with $\gamma = 0.98$), and increased learning rates for final states.

The resulting player was significantly better than a player that was trained by pure TD learning (without slope information or a predictive model). However, NeuroChess still was not a strong player, winning only 13% of its matches against GNUChess, even if the search depth of GNUChess was restricted to two half-steps. According to Thrun, “NeuroChess has learned successfully to protect its material, to trade material, and to protect its king. It has not learned, however, to open a game in a coordinated way, and it also frequently fails to play short endgames even if it has a material advantage. [...] Most importantly, it still plays incredibly poor openings, which are often responsible for a draw or a loss.”

17.2.2.2 KnightCap and TD-Leaf(λ)

The use of evaluation functions in *Chess* is usually combined with minimax search or some other multi-step lookahead algorithm. Such algorithms search the game tree up to some depth d (not necessarily uniformly over all branches), evaluate the deepest nodes with a heuristic evaluation function V and propagate the values up in the tree to the root. Of course, the evaluation function can be trained with TD or other RL methods. The naive implementation of temporal difference learning disregards the interaction with tree search: at time t , it tries to adjust the value of the current state, $V(x_t)$, towards $V(x_{t+1})$. However, the search algorithm does not use $V(x_t)$, but the values of states d steps down in the search tree. The idea of *TD-Leaf*(λ) is to use this extra information provided by the search tree.

To understand how TD-Leaf works, consider the game tree rooted at x_t , consisting of all moves up to depth d , with all the nodes evaluated with the heuristic function V . The *principal variation* of the tree is the path from the root where each player selects the minimax-optimal move with respect to V . Let x_t^ℓ denote the leaf node of the principal variation. TD-Leaf applies temporal difference learning to these principal leaves: it shifts $V(x_t^\ell)$ towards $V(x_{t+1}^\ell)$, using the TD(λ) update rule. This helps because the heuristic evaluation function is used exactly in these principal nodes (though in comparison to the other nodes). We note that TD-Leaf can be interpreted as multi-step TD-learning, where the multi-step lookahead is done by an improved policy.

KnightCap (Baxter et al, 2000) used *TD-Leaf*(λ) to train an evaluation function for *Chess*. The evaluation function is a linear combination of 5872 hand-coded

⁷ Quiescence search extends the search tree around “interesting” or “non-quiet” moves to consider their longer-term effects. For example, on the deepest level of the tree, capturing an opponent piece is considered very advantageous, but even a one-ply lookahead can reveal that it is followed by a counter-capture.

features. Features are divided into four groups, used at different stages of the game (opening, middle-game, ending and mating), and include piece material strengths, piece strengths on specific positions, and other standard features. The algorithm used eligibility traces with $\lambda = 0.7$ and a surprisingly high learning rate $\alpha = 1.0$, meaning that old values were immediately forgotten and overwritten. KnightCap attempted to separate the effect of the opponent: a positive TD-error could mean that the opponent made an error,⁸ so KnightCap made an update only if it could predict the opponent's move (thus it seemed like a reasonable move). Furthermore, for making learning progress, the material weights had to be initialized to their default values.

The key factor in KnightCap's success was the training regime. According to Baxter et al (2000), self-play converged prematurely and final performance was poor. Therefore, they let KnightCap play against humans of various levels on an internet chess server. The server usually matches players with roughly equal rankings, which had an interesting effect: as KnightCap learned, it played against better and better opponents, providing adequate experience for further improvement. As a result, KnightCap climbed from an initial Elo ranking of 1650 to 2150, which is slightly below the level of a human Master. With the addition of an opening book, Elo ranking went up to 2400-2500, a reasonable level.

We note that TD-Leaf has been applied successfully to other games, most notably, *Checkers* Schaeffer et al (2001), where the learned weight set was competitive with the best hand-crafted player at the time.

17.2.2.3 TreeStrap(minimax) and TreeStrap($\alpha\beta$)

Veness et al (2009) modified the idea of TD-Leaf: while TD-Leaf used the principal node of step t to update the principal node of step $t - 1$, TreeStrap(minimax) used it to update the search tree at time step t . Furthermore, not only the root node was updated, but every inner node of the search tree got updated with its corresponding principal node. TreeStrap($\alpha\beta$) used the same trick, but because of the pruning, the value of lower nodes was not always determined, only bounded in an interval. TreeStrap($\alpha\beta$) with expert trainers reached roughly the same level as TD-Leaf, but without needing its tricks for filtering out bad opponent moves and proper initialization. More importantly, TreeStrap learned much more efficiently from self-play. While expert-trained versions of TreeStrap still converged to higher rankings than versions with self-play, the difference shrunk to ≈ 150 Elo, weakening the myth that self-play does not work in *Chess*. There is anecdotal evidence that TreeStrap is able to reach ≈ 2800 Elo (Veness, personal communication).

The following observation gives the key to the success of TreeStrap: during the lookahead search, the algorithm will traverse through all kinds of improbable states (taking moves that no sane chess player would take). Function approximation has to make sure that these insane states are also evaluated more-or-less correctly (so that their value is lower than the value of good moves), otherwise, search is misled

⁸ We note that it could also mean an approximation error.

by the value function. As an additional factor, TreeStrap also uses resources more effectively: for each tree search, it updates every inner node.

17.2.2.4 Why Is *Chess* Hard to Reinforcement-Learn?

We could argue that the performance of reinforcement learning on *Chess* is not bad at all. After all, *Chess* has been one of the most actively researched games for decades, and RL has to compete with tree-search algorithms that are specialized to two-player, full-information, zero-sum games. In addition, lot of effort has been spent to tweak these search algorithms to perform well on *Chess*, and to build huge opening books. In contrast, RL approaches like TD-learning are more generic (e.g., able to handle randomness), so necessarily less effective. Still, it is worth asking the question why could RL not repeat its success in *Backgammon*.

For value-function-based methods, one of the reasons is that creating good features for *Chess* is not easy. Most heuristic state evaluation functions use features like piece values (the general value of a piece relative to a pawn), piece-square value (the worth of pieces on each specific field of the chessboard), features encoding pawn structure, defence of the king and mobility. These features are insufficient to discriminate between states with similar arrangement but radically different value. Thrun brings “knight fork” (Fig. 17.3) as an example, where the knight threatens the king and queen at the same time. To learn that the knight fork is dangerous (and differentiate it from positions where the knight is harmless), the agent has to learn about relative positions of pieces, and generalize over absolute positions, which is a quite complex task with a square-based representation.⁹

Few attempts have been made to learn such patterns. Variants of the *Morph* system (Gould and Levinson, 1992; Levinson and Weber, 2001) extract local patterns, and learn their value and relevance by a system that incorporates TD-learning. *Morph* is reported to be stronger than *NeuroChess*, but is still rather weak. Although not an RL work, the approach of Finkelstein and Markovitch (1998) is also worth mentioning. Instead of board patterns, they generate *move patterns* – in RL terms, we could call them state features and state-action (or state-macro) features. Move patterns can be used to filter or bias move selection, though no experiments were carried out to analyze the effect on playing strength.

Interestingly, *Chess* is also hard for state-space sampling methods (like UCT and other Monte-Carlo tree-search algorithms), even when augmented with heuristic evaluation functions. We delay the discussion of possible reasons to section 17.2.3.4.

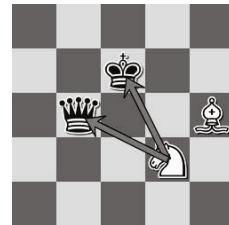


Fig. 17.3 Knight fork: the knight is threatening both the king and the queen

⁹ We note that this specific scenario can be resolved by quiescence search.

The deterministic nature of *Chess* is another factor that makes it hard for RL. The agents have to explore actively to get experience from a sufficiently diverse subset of the state space. The question of proper exploration is largely open, and we know of no *Chess*-specific solution attempt; though the question is studied abstractly in evolutionary game theory (Fudenberg and Levine, 1998). Additionally, determinism makes learning sensitive to the opponent’s strategy: if the opponent’s strategy is not sufficiently diverse (as it happens in self-play), learning will easily get stuck in an equilibrium where one player’s strategy has major weaknesses but the other player fails to exploit them.

While reinforcement learning approaches are not yet competitive for the control task, that is, *playing chess*, they are definitely useful for evaluation. Beal and Smith (1997) and later Droste and Fürnkranz (2008) applied TD-Leaf to learn the material values of pieces, plus the piece-square values. The experiments of Droste and Fürnkranz (2008) showed that the learned values performed better than the expert values used by several chess programs. The effect was much stronger in the case of non-conventional *Chess* variants like *Suicide chess*, which received much less attention in AI research, and have less polished heuristic functions.

17.2.3 Go

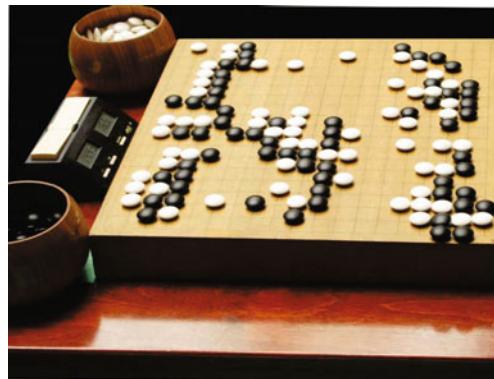


Fig. 17.4 19×19 Go board with a game in progress

way below the level of best human players. Because of its difficulty and popularity, *Go* took over the role of ‘Grand Challenge’ from *Chess*. While *Go* is a two-player, full information, zero sum game just like *Chess*, it is much harder for traditional minimax search-based methods (and others), for two major reasons. Firstly, the branching factor is huge: most empty board positions are legal moves. In addition,

Go is an ancient game of two players. On a 19×19 board, the players take turns to place stones (see Fig. 17.4). The overall goal is to conquer a larger territory than the opponent, by either occupying or surrounding it; groups of stones that are completely surrounded ‘die’ and are taken off from the board.¹⁰ Because of its difficulty, many programs play on smaller and simpler 9×9 or 13×13 boards.

To date, the best *Go* programs play on a Master level,

¹⁰ For a detailed description of the rules, see e.g.,
<http://senseis.xmp.net/?RulesOfGoIntroductory>.

humans can easily look ahead 60 or more half-moves in special cases (Schraudolph et al, 2001). Current tree search algorithms cannot go close to this depth yet. Secondly, it is hard to write good heuristic evaluation functions for *Go*. In *Chess*, material strengths and other similar measures provide a rough evaluation that is still good enough to be used in the leaf nodes. In *Go*, no such simple metrics exist (Müller, 2002), and even telling the winner from a *final* board position is nontrivial.

17.2.3.1 Features, Patterns

Evaluation functions in themselves are insufficient for guiding decision making in *Go*, but still useful to bias search. Frequently used features for candidate moves include distance to the previous move, to the border, capturing potential and others, see e.g., Table 1 of Coulom (2007) for more examples. These simple features are unable to capture the shapes of territories, which are essential in determining the value of *Go* states. Several approaches use *patterns* to improve evaluation, try to extract relevant patterns and/or learn their values.

Schraudolph et al (2001) uses TD-learning for a neural network that evaluates local patterns around candidate moves. Among other tricks, they exploit a special property of *Go*: Firstly, the final board state is much more informative than a single win/loss information: for each position, its contribution to the score is known (± 1 depending on the territory it belongs to plus points for each stone removed from there). The network can therefore try to predict the local reward for each position, using one output neuron per position. Dahl (2001) also uses neural networks to predict final territories, and another one to predict the *safety* of groups (whether the group will be dead or alive in the end). The networks are used in combination with $\alpha\beta$ -pruning in a complex decision system. It is not clear how much the learned extra information helps: most of the learned information may be superfluous.

Gelly and Silver (2008); Silver et al (2008) TD-learn the values of all local patterns up to size 3×3 . They use the learned evaluation function to guide search (see below). Coulom (2007) extracts frequently occurring patterns from recorded games of human experts. The evaluation of patterns is nonconventional and quite interesting: each candidate move is considered as a team of patterns (and features). The winner of the competing pattern teams is the one that corresponds to the move actually taken in the recorded game. A generalized Elo-rating-like formula is used then to assign a rating to each individual pattern and feature.

17.2.3.2 Dyna-2

The Dyna-2 architecture of Silver et al (2008) applies TD-learning in an interesting manner. The global evaluation function inevitably averages the value of many different local situations, overvaluing some bad cases and undervaluing some good ones. On the other hand, a locally fitted evaluation function is more accurate in that situation (that game) where it was trained, but does not generalize. The basic idea in Dyna-2 is to maintain both a *permanent memory*, which stores average feature weights over many games, and learnt by $TD(\lambda)$, and a *transient memory* which

modifies weights based on the current game situation, also trained by self-play $\text{TD}(\lambda)$. Transient memory is erased after each move selection. The function approximator for both components are linear combinations of several basic features plus shape patterns up to 3×3 for each board position (resulting in an impressive number of parameters to learn). The combination of transient and permanent memories performed better than either of them alone, and *Dyna-2* reached good performance on small boards.

17.2.3.3 UCT and Other Monte-Carlo Tree Search Methods

As a solution approach to the lack of good evaluation functions, Monte-Carlo (MC) sampling methods were proposed for heuristic board evaluation: from a given position, the evaluator finishes the game with random moves and gets the final outcome. The average of many such *rollouts* gives a measure of the value of the state. Bouzy and Helmstetter (2003) gives a nice overview of classical MC techniques in *Go*. Monte-Carlo value estimation is a well-studied area of RL, and soon advanced methods based on multi-armed bandits and sparse sampling were extended to game tree search. This brought a breakthrough in computer *Go*.

Traditional MC evaluation with tree search can be viewed as a search tree of two parts. The upper part is a full search tree of depth d (though it does not have to be of uniform depth) where node values are updated by the minimax rule; while the lower part is incomplete (usually not even stored in memory), consists of random paths to the final states and node values are the average of their children. Recent Monte-Carlo tree search (MCTS) algorithms remove the sharp boundary, the search tree can become strongly unbalanced and the update rule moves gradually from averaging to minimax.

The first such algorithm was *UCT* (Kocsis and Szepesvári, 2006), *upper confidence bounds applied to trees*, which is based on an efficient online learning algorithm for multi-armed bandits, UCB1 (Auer et al, 2002). For each node of the tree, UCT keeps track of the average payoff of previously seen rollouts from that node. If the number of samples is low, then the sample average can be quite inaccurate, so UCT also calculates a high-probability confidence interval of the payoff. Action selection is done optimistically: let $n(x)$ be the number of visits to the node for state x (the number of simulated games in which the corresponding state occurred), $n(x,a)$ the number of times when move a was selected, and $m(x,a)$ be the number of wins out of these. The algorithm selects the move with highest upper confidence bound

$$Q(x,a) = \frac{m(x,a)}{n(x,a)} + C \sqrt{\frac{\log n(x)}{n(x,a)}}, \quad (17.1)$$

where C is a suitably large constant, usually tuned experimentally, and generally much lower than the (conservative) value suggested by theory. In practice, the selection strategy is used only if there have been at least 50-100 visits to the state,

otherwise cheap random move selection is used.¹¹ Furthermore, to save storage, the tree is extended only with the first (several) nodes from the random part of the rollout. If n_i are low then UCT selects each move roughly equally often, while after lots of trials the bonus term becomes negligible and the minimax move is selected.

While the UCT selection formula has theoretical underpinnings, other formulas have been also tried successfully. Chaslot et al. suggest that the $O(1/\sqrt{n(x,a)})$ bonus is too conservative, and $O(1/n(x,a))$ works better in practice (Chaslot et al, 2008). In Chaslot et al (2009) they give an even more complex heuristic formula that can also incorporate domain knowledge. Coulom (2006) proposed another MCTS selection strategy that is similar in effect (goes gradually from averaging to minimax), but selection is probabilistic.

Although UCT or other MCTS algorithms are the basis of all state-of-the art *Go* programs, the playing strength of vanilla UCT can be improved greatly with heuristics and methods to bias the search procedure with domain knowledge.¹² Below we overview some of the approaches of improvement.

Biassing tree search. Gelly and Silver (2008) incorporate prior knowledge by using a heuristic function $Q^h(x,a)$. When a node is first added to the tree, it is initialized as $Q(x,a) = Q^h(x,a)$ and $n(x,a) = n^h(x,a)$. The quantity n^h indicates the confidence in the heuristic in terms of equivalent experience. The technique can be alternatively interpreted as adding a number of virtual wins/losses (Chatrion et al, 2008; Chaslot et al, 2008). Progressive unpruning/widening (Chaslot et al, 2008; Coulom, 2006) severely cuts the branching factor: initially, only the heuristically best move candidates are added to each node. As the node is visited more times, more tree branches are added, corresponding to move candidates that were considered weaker by the heuristics.

RAVE (Gelly and Silver, 2008, rapid action value estimation), also called “all-moves-as-first” is a rather *Go*-specific heuristics that takes *all* the moves of the player along a game and update their visit counts and other statistics. The heuristic treats actions as if their order did not matter. This gives a quick but distorted estimate of values.

Biassing Monte-Carlo simulations. It is quite puzzling why Monte-Carlo evaluation works at all: the simulated random games correspond to a game between two extremely weak opponents, and it is not clear why the evaluation of states is useful against stronger opponents. It seems reasonable that the stronger the simulation policy is, the better the results. This is true to some extent, but there are two opposing factors. Firstly, simulation has to be diverse for sufficient exploration; and secondly, move selection has to be very light on computation, as we need to run it millions of times per game. Therefore, the rollout policy is critical to the success of

¹¹ The switch from UCT move selection to random selection can also be done gradually. Chaslot et al (2008) show that this can increase efficiency.

¹² See <http://senseis.xmp.net/?CGOSBasicUCTBots> and <http://cgos.boardspace.net/9x9/allTime.html> for results of pure UCT players on the Computer *Go* Server. The strongest of these has an Elo rating of 1645, while the strongest unconstrained programs surpass 2500.

MCTS methods. Gelly et al (2006) uses several 3×3 patterns that are considered interesting, and chooses moves that match the patterns with higher probability. Most other *Go* programs use similar biasing.

Silver and Tesauro (2009) suggested that the weakness of the rollout play is not a problem as long as opponents are *balanced*, and their errors somehow cancel out. While the opponents are equally good on average (they are identical), random fluctuations may imbalance the rollout. Silver and Tesauro (2009) propose two methods to learn a MC simulation policy with low imbalance, and they show that it significantly improves performance.

17.2.3.4 Why Does MCTS Work in *Go*?

The success of UCT and other MCTS methods in *Go* inspired applications to many other games in a wide variety of genres, but its success was not repeated in each case—most notably, *Chess* (Ramanujan et al, 2010). The question naturally arises: why does UCT and other MCTS methods work so well in *Go*? From a theoretical perspective, Coquelin and Munos (2007) showed that in the worst case, UCT can perform much worse than random search: for finding a near-optimal policy, the required number of samples may grow as a nested exponential function of the depth. They also propose an alternative BAST (bandit algorithm for smooth trees) which has saner worst-case performance. Interestingly, the better worst-case performance of BAST does not lead to a better performance in *Go*. The likely reason is that less conservative update strategies are better empirically (Chaslot et al, 2008), and BAST is even more conservative than UCT.

Policy bias vs. Value bias. According to Szepesvári (2010, personal communication), success of MCTS can (at least partly) attributed to the fact that it enables a different bias than value-function-based methods like TD. Function approximation for value estimation puts a “value bias” on the search. On the other hand, the heuristics applied in tree search methods bias search in policy space, establishing a “policy bias”. The two kinds of biases have different strengths and weaknesses (and can also be combined, like Dyna-2 did), and for *Go*, it seems that policy bias is easier to do well.

Narrow paths, shallow traps. UCT estimates the value of a state as an average of outcomes. The average concentrates around the minimax value, but convergence can take a long time. (Coquelin and Munos, 2007) Imagine a hypothetical situation where Black has a “narrow path to victory”: he wins if he carries out a certain move sequence exactly, but loses if he makes even a single mistake. The minimax value of the state is a ‘win’, but the average value will be closer to ‘lose’ unless half of the weights is concentrated on the winning branch. However, the algorithm has to traverse the full search tree before it can discover this.

In the same manner, averaging can be deceived by the opposite situation where the opponent has a winning strategy, but any other strategy is harmless. Unless we leave MCTS a very long time to figure out the situation, it will get trapped and think that the state is safe. This is especially harmful if the opponent has a short

winning strategy from the given state: then even a shallow minimax search can find that winning strategy and defeat MCTS. Ramanujan et al (2010) call these situations “shallow traps”, and argue that (tragically) bad moves in *Chess* often lead to shallow traps, making *Chess* a hard game for MCTS methods.

Narrow paths and shallow traps seem to be less prevalent in *Go*: bad moves lead to defeat many steps later, and minimax-search methods cannot look ahead very far, so MCTS can have the upper hand. Nevertheless, traps do exist in *Go* as well. Chaslot et al (2009) identify a situation called “nakade”, which is a “situation in which a surrounded group has a single large internal, enclosed space in which the player won’t be able to establish two eyes if the opponent plays correctly. The group is therefore dead, but the baseline Monte-Carlo simulator sometimes estimates that it lives with a high probability”. Chaslot et al (2009) recognize and treat nakade separately, using local minimax search.



Fig. 17.5 Screenshot of Tetris for the Nintendo entertainment system (NES). The left-hand column lists the seven possible tetris pieces. The standard names for the tetromino are in order: T, J, Z, O, S, L, I.

17.2.4 Tetris

Tetris is a computer game created by Alexey Pajitnov in 1984, played on a 10×20 board (see Fig. 17.5). Tetrominoes fall down the board one by one, each new tetromino is drawn uniformly at random. The player can control the place and rotation where the current tetromino falls. The game ends when the board fills up, but full lines are cleared. Points are scored for clearing lines (and other things); the goal of the game is to maximize score. In a more abstract variant, used widely in AI

research (Bertsekas and Tsitsiklis, 1996), pieces do not fall, the player just needs to give the rotation and column where the current piece should fall. Furthermore, scoring is flat, each cleared line is worth +1 reward.¹³ Many variants exist with different board size, shape set, distribution of shapes.

As a fully-observable, single-player game with randomness, *Tetris* fits the MDP framework well, where most of RL research is concentrated. Two other factors helped its popularity: its simplicity of implementation and its complexity of optimal control. Optimal placement of tetrominoes is NP-hard even to approximate, even if the tetromino sequence is known in advance. Therefore, *Tetris* became the most popular game for testing and benchmarking RL algorithms.

Besides the properties that make *Tetris* a good benchmark for RL, it has the unfortunate property that the variance of a policy's value is huge, comparable to its mean. In addition, the better a policy is, the longer it takes to evaluate it (we have to play through the game). This makes it hard to compare top-level algorithms or investigate the effects of small changes. For these reasons, Szita and Szepesvári (2010) proposed to use the variant *SZ-Tetris* that only uses the ‘S’ and ‘Z’ tetrominoes. They argue that this variant is the “hard core” of *Tetris*, that is, it preserves (even amplifies) most of the complexity in Tetris, and maximum scores (and variances) are much lower and seems therefore a much better experimental testbed.

Tetris is also special because it's one of the few applications where function approximation of the value function does not work well. Many of the value-function-approximating RL methods are known to diverge or have only weak performance bounds. However, theoretical negative results (or lack of positive results) are often dismissed as too conservative, as value function approximation generally works well in practice. *Tetris* is a valuable reminder that the theoretical performance bounds might not be so loose even for problems of practical interest.

Bertsekas and Tsitsiklis (1996) were among the first ones to try RL on Tetris. States were represented as 21-element feature vectors consisting of the individual column heights (10 features), column height differences (9 features), maximum column height and the number of holes. This representation with linear function approximation served as a basis for most subsequent approaches,¹⁴ which provides an excellent opportunity to compare *value-function-based* and *preference-function-based* methods. In both groups, a linear combination of feature functions, $V(x) = \sum_{i=1}^k w_i \phi_i(x)$ is used for decision making, but in the first group of methods, $V(x)$ tries to approximate the optimal value function $V^*(x)$, while in the second group no attempt is made to ensure $V(x) \approx V^*(x)$, making the parameter optimization less constrained. Preference functions can also be considered a special case of direct policy search methods.

¹³ For detailed rules of original *Tetris*, please see section 5 of http://www.colinfahey.com/tetris/tetris_en.html; for the specification of the abstract version, cf. Bertsekas and Tsitsiklis (1996).

¹⁴ Though many other features were designed and tried; Table 1 of Thiery and Scherrer (2009) summarizes these.

17.2.4.1 Value-Function-Based Methods

To learn the weights w_i , Bertsekas and Tsitsiklis (1996) apply λ -policy iteration, which can be considered as the planning version of TD(λ), and generalizes value- and policy-iteration. The best performance they report is around 3200 points, but performance gets worse after training continues. Lagoudakis et al (2002) tries least-squares policy iteration; Farias and van Roy (2006) applies approximate linear programming with an iterative process for generating samples; Kakade (2001) applies natural policy gradient tuning of the weights. The performance of these methods is within the same order of magnitude, in the range 3000–7000.

17.2.4.2 Preference-Function-Based Methods

Böhm et al (2004) tune the weight vector with an evolutionary algorithm, using two-point crossover, Gaussian noise as the mutation operator, and a random weight rescaling operator which did not affect the fitness of the weight vector, but altered how it reacted to future genetic operations. Szita and Lörincz (2006a) applied the cross-entropy method (CEM), which maintains a Gaussian distribution over the parameter space, and adapts the mean and variance of the distribution so as to maximize the performance of weight vectors sampled from the distribution, while maintaining a sufficient level of exploration. Their results were later improved by Thiery and Scherrer (2009) using additional feature functions, reaching 35 million points. While CEM draws each vector component from an independent Gaussian (that is, its joint distribution is an axis-aligned Gaussian), the CMA-ES method (covariance matrix adaptation evolutionary strategy) allows general covariance matrices at the cost of increasing the number of algorithm parameters. Boumaza (2009) apply CMA-ES to Tetris, reaching results similar to Thiery and Scherrer (2009).

17.2.4.3 Preference Functions vs. Approximate Value Functions and the Role of Features in Tetris

Given an approximate value function V , $V(x)$ is approximately the total reward collectible from x . Preference functions do not have this extra meaning, and are only meaningful in comparison: $V(x) > V(y)$ means that the algorithm prefers x to y (probably because it can collect more reward from x). Infinitely many preference functions induce the same policy as a given value function, as the policies induced by preference functions are invariant to rescaling or any other monotonously increasing transformations. The exact solution to the Bellman optimality equations, V^* , is also an optimal preference function, so solving the first task also solves the other. In the function approximation case, this is not true any more, a value function that is quite good in minimizing the approximate Bellman error might be

very bad as a preference function (it may rank almost all actions in the wrong order, while still having a small Bellman residual). Furthermore, the solution of the approximate Bellman equations does not have a meaning – in fact, it can be arbitrarily far from V^* . Usually this happens only in artificial counterexamples, but weak results on *Tetris* with its traditional feature representation indicate that this problem is also subject to this phenomenon. Szita and Szepesvári (2010) One way to avoid problems would be to change the feature functions. However, it is an interesting open question whether it's possible to unify the advantages of preference functions (direct performance optimization) and approximate value functions (the approximate Bellman equations allow bootstrapping: learning from $V(x_{t+1})$).

17.2.4.4 How Good Are RL Algorithms in Tetris?

In certain aspects, *Tetris*-playing RL agents are well beyond human capabilities: the agent of Thiery and Scherrer (2009) can clear 35 million lines on average. Calculating with a speed of 1 tetromino/second (reasonably good for a human player), this would take 2.77 years of nonstop playing time. However, observation of the game-play of AI shows that they make nonintuitive (and possibly dangerous) moves that a human would avoid. Results on SZ-Tetris also support the superiority of humans so far: the CEM-trained controller reaches 133 points, while a simple hand-coded controller, which would play weaker than a good human player, reaches 182 points (Szita and Szepesvári, 2010).

17.2.5 Real-Time Strategy Games

Games in the real-time strategy (RTS) genre are war-simulations. In a typical RTS (Fig. 17.6), players need to gather resources, use them for building a military base, for technological development and for training military units, destroying bases of the opponents and defend against the opponents' attacks. Tasks of the player include ensuring a sufficient flow of resource income, balancing resource allocation between economical expansion and military power, strategic placement of buildings and defenses, exploration, attack planning, and tactical management of units during battle. Due to the complexity of RTS games, RL approaches usually pick one or several subtasks to learn while relying on default heuristics for the others. Listings of subtasks can be found in Buro et al (2007); Laursen and Nielsen (2005); Ponsen et al (2010). One of the distinctive properties of RTS games, not present in any of the previously listed games, is the need to handle parallel tasks. This is a huge challenge, as the parallel tasks interact, and can span different time scales.

Most implementations use the *Warcraft* and *Starcraft* families, the open-source RTS family *Freecraft/Wargus/Stratagus/Battle of Survival/BosWars*,¹⁵ or the *ORTS*

¹⁵ For connections within this many-named family, see <http://en.wikipedia.org/wiki/Stratagus#History>.



Fig. 17.6 Screenshot of *BosWars*, an open-source real-time strategy game

game engine which was written specifically to foster research (Buro and Furtak, 2003). The variety of tasks and the variety of different RTS games is accompanied by a colorful selection of RL approaches.

17.2.5.1 Dynamic Scripting

Dynamic scripting (DS) was originally applied to tactical combat in role-playing games Spronck et al (2006), but application to RTS soon followed. Dynamic scripting assembles policies by randomly drawing K rules from a pool of rules. The probability of selecting a rule is proportional to its weight, initially all weights are equal. The drawn policies are tested in-game, and its rules are reinforced depending on the outcome: weights are increased if the policy was victorious and decreased otherwise. The rest of the rule weights is adjusted as well, to keep the total weight constant, and minimum/maximum weights may be prescribed to maintain sufficient exploration. Ponsen and Spronck (2004) implemented DS in *Wargus*, by learning separate scripts for each abstract state of the game. States are defined by the set of buildings in the agent's base (which largely determines the set of available actions). Scripts in each state were rewarded individually, using a combination of the final outcome and a heuristic fitness evaluation at the time of leaving the state. The rules of a script belong to one of four categories, build, research, resource, or combat. Rules in different categories can run in parallel. Dynamic scripting learned to play better than the default AI and most hand-coded AIs (though not “Knight’s rush”, a strategy that is considered near-optimal). The fitness function used by Ponsen and Spronck (2004) used a simple material strength evaluation (similar to the one

used in *Chess*). Kerbusch (2005) has shown that TD-learning can do a better job in tuning the relative strengths of units than hand-coding, and the updated evaluation function improves the playing strength of dynamic scripting. Kok (2008) used an implicit state representation: the preconditions of each rule determined whether the rule is applicable; furthermore, the number of applicable actions was much lower, typically around five. With these modifications, he reached good performances both with vanilla DS and Monte-Carlo control (equivalent to TD(1)) with ε -greedy exploration and a modification that took into account the fixed ordering of rules.

17.2.5.2 Evolutionary Learning

Ponsen and Spronck (2004) also tried evolutionary learning on *Wargus*, with four genes for each building-state, corresponding to the four action categories. In a follow up paper, Ponsen et al (2005) get improved results by pre-evolving the action sets, by selecting several four-tuples of actions per state that work well in conjunction.

EvolutionChamber is a recent, yet unpublished approach¹⁶ to learn opening moves for *Starcraft*. It uses an accurate model of the game to reach a predefined goal condition (like creating at least K units of a type) as quickly as possible. Evolution-Chamber works under the assumption that the opponent does not interact, and needs a goal condition as input, so in its current form it is applicable only to openings. As a huge advantage, however, it produces opening strategies that are directly applicable in *Starcraft* even by human players, and it has produced at least one opening (the “seven-roach rush”, getting 7 heavily armored ranged units in under five minutes) which was previously unknown and is regarded very effective by expert players.¹⁷

17.2.5.3 Case-Based Learning

In case-based reasoning, the agent stores several relevant situations (cases), together with policies that tell how to handle those. Relevant cases can be added to the case base either manually Szczępański and Aamodt (2009) or automatically (either offline or online). When the agent faces a new situation, it looks up the most similar stored case(s) and makes its decision based on them, and (when learning is involved) their statistics are updated based upon the reinforcement. The abstract states, set of actions and evaluate function used by Aha et al (2005) are identical to the ones used by Ponsen and Spronck (2004), distance of cases is determined from an eight-dimensional feature vector. If a case is significantly different from all stored cases, it is added to the pool, otherwise, updates the reward statistics of existing cases. The

¹⁶ See <http://lbrandy.com/blog/2010/11/using-genetic-algorithms-to-find-starcraft-2-build-orders/> for detailed description and <http://code.google.com/p/evolutionchamber/> for source code. We note that even though the algorithm is not published through an academic journal or proceedings, both the algorithm and the results are publicly available, and have been verified by experts of the field.

¹⁷ See forum discussion at http://www.teamliquid.net/forum/viewmessage.php?topic_id=160231.

resulting algorithm outperformed the AIs evolved by Ponsen and Spronck (2004), and is able to adapt to several unknown opponents (Molineaux et al, 2005). Weber and Mateas (2009) introduce an improved distance metric similar to the “edit distance”: they define a distance of two cases as the resource requirement to transfer from one to the other.

17.2.5.4 How Good Are RL Approaches in Real-Time Strategies?

The listed approaches deal with high-level decision making, delegating lower-level tasks like tactical combat or arrangement of the base to default policies. Disregarding the possible interactions certainly hurts their performance, e.g., fewer soldiers may suffice if their attack is well-coordinated and they kill off opponents quickly. But even discounting that, current approaches have apparent shortcomings, like the limited ability to handle parallel decisions (see Marthi et al (2005) for a solution proposal) or the limited exploration abilities. For example, Ponsen and Spronck (2004) report that late-game states with advanced buildings were rarely experienced during training, resulting in weaker strategies there.

State-of-the-art computer players of RTS are well below the level of the best human players, demonstrated, for example, by the match between *Overmind*, winner of the 2010 AIIDE Starcraft Competition and retired professional player =DoGo=. As of today, top-level AIs for real-time strategies, for example, the contest entries of the StarCraft competitions and ORTS competitions, do not use the reinforcement techniques listed above (or others).

17.3 Challenges of Applying Reinforcement Learning to Games

The previous section explored several games and solution techniques in depth. The games were selected so that they each have several interesting RL issues, and together they demonstrate a wide scale of issues RL has to face. By no metric was this list representative. In this section we give a list of important challenges in RL that can be studied within games.

17.3.1 Representation Design

As it has already been pointed out in the introduction of the book, the question of suitable representation plays a central role in all applications of RL. We usually have rich domain knowledge about games, which can aid representation design. Domain knowledge may come in many forms:

- Knowledge of the rules, the model of the game, knowing what information is relevant;
- Gameplay information from human experts. This may come in the form of advice, rules-of-thumb, relevant features of the game, etc.;

- Databases of recorded games, opening books, endgame databases;
- Rich evaluative feedback. In many cases, the agent gets more evaluative information than just the reward signal (but less than full supervision), like a decomposition of the reward with respect to some subgoals. For example, in a combat of teams, besides the global reward (victory/defeat), each member gets individual feedback (remaining hit points, amount of harm inflicted/prevented).

Consequently, in game applications we can find a wide variety of methods to incorporate domain knowledge in the representation, together with approaches that try to obtain representations with as little domain knowledge as possible.

17.3.1.1 Feature Extraction, Abstraction

Extracting relevant information from raw sensory information is an essential part of obtaining the representation. Raw observations are usually assumed to be in a somewhat preprocessed form; for example, extracting state information from screen captures is usually not considered as a part of the representation design problem (and delegated to other areas of machine learning, like machine vision). We mention two interesting exceptions: (1) Entrants of the Ms. Pac-Man Competition¹⁸ need to write controllers using 15 frames/second video capture as inputs, which is a far from trivial task even with a very accurate model of the game. (2) Naddaf (2010) tries to learn Atari 2600 games from their memory dump (including video information).¹⁹

Feature mappings of the input play two roles (not independent from each other): they abstract away less relevant information, and map similar situations to similar feature vectors, promoting generalization. The meaning of “similarity” depends largely on the architecture that is fed by the feature vectors. In most cases, the architecture aims to approximate the value function, though model-based and/or structured (relational, factored) RL methods can also be utilized. Function approximation may be linear or nonlinear, in the latter case, neural networks are the most common choice.

Below we list several possible types of representations. Their usage (either with model-free function approximation or in a model-based architecture) is treated in detail in other parts of the book, so here we make only one remark: most of the algorithms have no guarantees of convergence when used with function approximation, yet used in applications widely, and without any problems reported. We note

¹⁸ See <http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html>. The contest is coordinated by Simon Lucas, and has been organized on various evolutionary computing and game AI conferences (WCCI, CEC and GIG).

¹⁹ The Atari 2600 system is an amazing combination of simplicity and complexity: its memory, including the video memory, is 1024 bits, including the video output (that is, the video had to be computed while the electron ray was drawing it). This amount of information is on the verge of being manageable, and is a Markovian representation of the state of the computer. On the other hand, the Atari 2600 was complex (and popular) enough so that it got nearly 600 different games published.

that part of the reason is definitely ‘‘presentation bias’’: if an application did not work because of parameter sensitivity / divergence / too slow convergence / bad features, its probability of publication is low.

Tabular representation. For games with discrete states, the trivial feature mapping renders a unique identifier to each state, corresponding to ‘‘tabular representation’’. The performance of dynamic programming-based methods depends on the size of the state space, which usually makes the tabular representation intractable. However, state space sampling methods like UCT remain tractable, as they avoid the storage of values for each state. They trade off the lack of stored values and the lack of generalization by increased pre-movement decision time. For examples, see applications to *Go* (section 17.2.3), *Settlers of Catan* (Szita et al, 2010), or general game playing (Björnsson and Finnsson, 2009).

Expert features. Game situations are commonly characterized by several numerical quantities that are considered relevant by an expert. Examples to such features are material strength in *Chess*, number of liberties in *Go*, or the number of holes in *Tetris*. It is also common to use binary features like the presence of a pattern in *Chess* or *Go*.

Relational representation. Relational (Džeroski et al, 2001), object-oriented (Diuk et al, 2008) and deictic (Ponsen et al, 2006) representations are similar, they assume that the game situation can be characterized by several relevant relationships between objects (technically, a special case of expert features, and each relationship involves a few objects only). Such representations may be used either directly for value function approximation (Szita and Lörincz, 2006b), or by some structured reinforcement learning algorithm (see Chapters 8 & 9)

State aggregation, clustering. While in some cases it is possible to enumerate and aggregate ground states directly (Gilpin et al, 2007), it is more common to perform clustering in some previously created feature space. Cluster centres can be used either as cases in case-based reasoning or as a high-level discretization of the state space.

Combination features. Conjunction features require the combined presence of several (binary) features, while more generally, we can take the combination of features via any logical formula. Combination features are a simple way of introducing non-linearity in the inputs, so they are especially useful for use with linear function approximation. Combination features are also a common subject of automatic representation search techniques (see below.)

17.3.1.2 Automatic Feature Construction

Hidden layers of neural networks define feature mappings of the input space, so methods that use neural networks can be considered as implicit feature construction methods. Tesauro (1995) reports that TD-Gammon learned interesting patterns, corresponding to typical situations in *Backgammon*. Moriarty and Miikkulainen (1995)

use artificial evolution to develop neural networks for *Othello* position evaluation, and manage to discover advanced *Othello* strategies.

The Morph system of Gould and Levinson (1992) extracts and generalizes patterns from chess positions, just like Levinson and Weber (2001). Both approaches learn the weights of patterns by $\text{TD}(\lambda)$. Finkelstein and Markovitch (1998) modify the approach to search for state-action patterns. The difference is analogous to the difference of the value functions $V(x)$ and $Q(x,a)$: the latter does not need lookahead for decision making.

A general approach for feature construction is to combine atomic features into larger conjunctions or more complex logical formulas. The Zenith system of Fawcett and Utgoff (1992) constructs features for *Othello* from a set of atomic formulas (of first-order predicate calculus), using decomposition, abstraction, regression, and specialization operators. Features are created iteratively, by adding features that help discriminating between good and bad outcomes. The features are pruned and used in a linear evaluation function. A similar system of Utgoff and Precup (1998) constructs new features in *Checkers* by the conjunction of simpler boolean predicates. Buro (1998) uses a similar system to create conjunction features for *Othello*. For a good summary on feature construction methods for games, see Utgoff (2001). More recent results on learning conjunction features include Shapiro et al (2003) for *Diplomacy*, Szita and Lörincz (2006b) for *Ms. Pac-Man* and Sturtevant and White (2007) for *Hearts*.

Gilpin and Sandholm (2007) study abstractions of *Poker* (and more generally, two-player imperfect-information games), and provide a state-aggregation algorithm that preserves equilibrium solutions of the game, that is, preserves relevant information with respect to the optimal solution. In general, some information has to be sacrificed in order to keep the size of the task feasible. Gilpin et al (2007) propose another automatic abstraction method that groups information states to a fixed number of clusters so that states within a cluster have roughly the same distribution of future rewards. Ponsen et al (2010) give an overview of (lossless and lossy) abstraction operators specific to real-time strategy games.

In general, it seems hard to learn representations without any domain knowledge (Bartók et al, 2010), but there are a few attempts to *minimize* the amount of domain knowledge. Examples can be found in the *general gameplay* literature, see e.g., Sharma et al (2009); Günther (2008); Kuhlmann (2010). These approaches look for general properties and invariants that are found in most games: variables of spatial/temporal type, counting-type variables, etc.

17.3.2 Exploration

Exploration is a central issue in driving the agent towards an optimal solution. In most game applications, the algorithm is expecting the game dynamics in backgammon or the opponent do the exploration (training against a variety of opponents), or just hope that all will be well. We will overview the choices and aspects of

opponent selection in Section 17.3.3, and concentrate here on the agent’s own role in enforcing variety.

Boltzmann and ϵ -greedy action selection are the most commonly occurring form of exploration. We know from the theory of finite MDPs that much more efficient exploration methods exist (Chapter 6), but it is not known yet how to extend these methods to model-free RL with function approximation (which covers large part of the game applications of RL). UCT and other Monte-Carlo tree search algorithms handle exploration in a somewhat more principled way, maintaining confidence intervals representing the uncertainty of outcomes, and apply the principle of “optimism in the face of uncertainty”. When a generative model of the game is available (which is the case for many games), even classical planning can simulate exploration. This happens, for example, in *TD-Leaf* and *TreeStrap* where limited lookahead search determines the most promising direction.

Evolutionary RL approaches explore the parameter space in a population-based manner. Genetic operators like mutation and crossover ensure that the agents try sufficiently diverse policies. RSPSA (Kocsis et al, 2006) also does exploration in the parameter space, but uses local search instead of maintaining a population of agents.

17.3.3 Source of Training Data

For games with two or more players, the environment of the learning agent depends on the other players. The goal of the learning agent can be to find a Nash-equilibrium, a best response play against a fixed opponent, or just performing well against a set of strong opponents. However, learning can be very inefficient or completely blocked if the opponent is much stronger than the actual level of the agent: all the strategies that a beginner agent tries will end up in a loss with very high probability, so the reinforcement signal will be uniformly negative – not providing any directions for improvement. In another note, opponents have to be sufficiently diverse to prevent the agent from converging to local optima. For these reasons, the choice of training opponents is a nontrivial question.

17.3.3.1 Self-play

Self-play is by far the most popular training method. According to experience, training is quickest if the learner’s opponent is roughly equally strong, and that definitely holds for self-play. As a second reason for popularity, there is no need to implement or access a different agent with roughly equal playing strength. However, self-play has several drawbacks, too. Theoretically, it is not even guaranteed to converge (Bowling, 2004), though no sources mention that this would have caused a problem in practice. The major problem with self-play is that the single opponent does not provide sufficient exploration. Lack of exploration may lead to situations like the

one described in Section 6.1.1 of Davidson (2002) where a Poker agent is trapped by a self-fulfilling prophecy: suppose that a state is (incorrectly) estimated to be bad. Because of that, the agent folds, loses the game, so the estimated value of the state will go further down.

17.3.3.2 Tutoring

As an alternative to self-play, the agent may be tutored by a set of different opponents with increasing strength. For more popular games (like *Chess* and *Go*), such a series of opponents can be found on game servers that match players based on playing strength. However, such a pool of players is not available for every game, and even if it is, games are typically much slower, reducing the number of games that can be played. On the positive side, agents trained by tutoring are typically significantly stronger than the ones trained by self-play, see, e.g., Baxter et al (2001); Veness et al (2009).

17.3.3.3 Other Training Strategies

Epstein (1994) proposed a “lesson and practice” setup, when periods of (resource-expensive) tutoring games are followed by periods of self-play. Thrun (1995) trained NeuroChess by self-play, but to enforce exploration, each game was initialized by several steps from a random game of a game database. This way, the agent started self-play from a more-or-less random position.

Learning from the observation of game databases is a cheap way of learning, but usually not leading to good results: the agent only gets to experience good moves (which were taken in the database games), so it never learns about the value of the worse actions and about situations where the same actions are bad.

17.3.4 Dealing with Missing Information

Agents may have to face missing information in both classical and computer games, for example, hidden cards in *Poker*, partial visibility in first-person shooters or “fog of war” in real-time strategies. Here we only consider information that is “truly missing”, that is, which could not be obtained by using a better feature representation. Furthermore, missing information about the opponents’ behavior is discussed separately in section 17.3.5.

One of the often-used approaches is to ignore the problem, and learn reactive policies based on current observation. While TD-like methods can diverge under partial observability, $\text{TD}(\lambda)$ with large λ (including Monte-Carlo estimation) is reported to be relatively stable. The applicability of direct policy search methods, including dynamic scripting (Spronck et al, 2006), cross-entropy policy search (Szita

and Lörincz, 2006b) and evolutionary policy search (Koza, 1992) is unaffected by partial observability.

Monte-Carlo sampling, however, becomes problematic in adversarial search: after sampling the unobserved quantities, the agent considers them observable, and also assumes that the opponent can observe everything. This way, decision making ignores the “value of information”, when a move is preferred because it gets extra information (or keeps information hidden from the opponent). Ginsberg (2002) introduces a variant of minimax search over *information sets*, sets of states that cannot be distinguished by the information the agent has. They apply a simplified version to the *Bridge*-playing program GIB. Amit and Markovitch (2006) use MC sampling, but for each player, information unknown to them is masked out. The technique is applied to improved bidding in *Bridge*.



Fig. 17.7 A game of Texas Hold 'em poker going on. Screenshot from *Hoyle's Texas Hold Em* for the Xbox 360.

The concept of *counterfactual regret* (CFR) (Zinkevich et al, 2008) is analogous to MCTS, but it cancels out the effect of the opponent from the value calculation, and is applicable to hidden-information games. The combination of CFR and a sophisticated abstraction of the state space is currently the state of the art in *Poker* (Risk and Szafron, 2010; Schnizlein et al, 2009).²⁰

17.3.5 Opponent Modelling

The behavior of the opponent is a kind of hidden information. Opponent models try to predict the moves of the opponent based on past observations, possibly from many previous games. Opponent modelling is an important augmentation of

²⁰ See Rubin and Watson (2011) for a comprehensive review on computer poker.

multiplayer RL systems, though it is usually not formalized as a task itself. In two-player zero-sum games, playing a minimax-optimal strategy, or, in the partially observable case, a Nash-equilibrium strategy, has guaranteed good performance even in the worst case – but actually, if the opponent is not perfect then exploiting their weaknesses can be even better. The Poker-playing system Vexbot (Billings et al, 2006) learns a model of the opponent and calculates the best-response policy. According to Lorenz (2004), opponent modelling makes sense even in perfect information games. He observed that in a tournament against strong but imperfect players, the minimax-optimal strategy does not have an overall good performance: it plays too conservatively and fails to exploit the mistakes of the other players. Davidson (2002) substitutes minimax-search with “Probimax” (also called Expectimax in earlier literature), which models the opponent’s strategy by a probability distribution, and selects the opponent’s moves in the tree according to this distribution. He demonstrates the utility of Probimax on *Chess*: in a state which is a game-theoretical loss, minimax considers each candidate move equally bad (all of them lead to a loss). Probimax, on the other hand, is able to differentiate between moves, and select the one where the opponent has the highest chance to err.

17.4 Using RL in Games

So far, the chapter has focused on the RL perspective: algorithmic issues and types of challenges characteristic to games. In this section, we take a look at the other side of the RL–games relationship: how can RL be useful in games. The two viewpoints more-or-less coincide when the goal is to find a strong AI player. However, RL in games can be useful in more roles than that, which is most apparent in modern video games. In this section we will overview how can reinforcement learning be of use in game development and gameplay.

17.4.1 *Opponents That Maximize Fun*

In contrast to games in AI research, the objective of a commercial video game is to keep the player entertained (Scott, 2002; Gilgenbach, 2006). Part of this task is to set the difficulty to the right level, neither too strong nor too easy, rewarding the player with a hard earned victory. In this respect, an ideal AI opponent is just slightly weaker than the player at his best. However, it is equally important that the game agent loses in a convincing way. According to Gilgenbach (2006), the agent needs to create the illusion that it’s losing despite it’s behaving intelligently and is doing its best to win. Of course, neither ‘fun’, nor ‘challenging’ and ‘intelligent’ are well-defined concepts, allowing many different interpretations. Below we review some that apply RL.

In plenty of games, making the computer smart enough is a challenge, but making it weak enough might be one, too, as the example of Pong shows. Pong was among the first video games with a computer opponent. The optimal policy is trivial (the paddle should follow the y coordinate of the ball), but obviously not many people would play Pong against the optimal policy. Imagine now an agent that plays perfectly 45% of the time and does nothing in the rest – matches against this agent would be close, but probably still not much fun. McGlinchey (2003) tries to extract believable policies from recordings of human players. Modelling human behavior is a popular approach, but is outside the scope of this chapter; see e.g., Hartley et al (2005) for an overview.

17.4.1.1 Difficulty Scaling

Difficulty scaling techniques try to tune the difficulty of the game automatically. They take a strong (possibly adaptive) strategy, and adjust its playing strength to the level of the player, if necessary.

Spronck et al (2004) modify dynamic scripting. In the most successful of the three modifications considered, they exclude rules that make the strategy too strong. They maintain a weight threshold W_{\max} that is lowered for any victory and increased in case of a defeat; if the weight of a rule exceeds W_{\max} , it cannot be selected. They applied the technique to tactical combat in a simplified simulation of the RPG *Baldur's Gate*, reaching reliable and low-variance difficulty scaling against a number of tactics. Andrade et al (2004) applied a similar technique to a Q-learning agent in a simplified fighting game *Knock'em*: if the agent was leading by too much, several of the best actions (ranked by Q-values) got disabled. Thanks to the use of Q-functions, difficulty adaptation could be done online, within episodes. The difference between hitpoints of players determined the leader and whether the game was deemed too difficult. Hunicke and Chapman (2004) propose a difficulty adjustment system that modifies the game environment, e.g., gives more weapons to the player and spawns fewer enemies when the game is considered too hard. A similar system was implemented in the first-person shooter *Max Payne*,²¹ where the health of enemies and the amount of auto-aim help are adjusted, based on the player's performance. Hagelbäck and Johansson (2009) aim at specific score difference in an RTS game. According to questionnaires, their method significantly raised enjoyment and perceived variability compared to static opponents.

17.4.1.2 In-Game Adaptation

Reinforcement learning can be used for online adaptation of opponents to the player's style of play. In principle, this can prevent the player from applying the same tricks over and over again, forcing a more diverse gameplay, and also

²¹ [http://www.gameontology.org/index.php/
Dynamic_Difficulty_Adjustment](http://www.gameontology.org/index.php/Dynamic_Difficulty_Adjustment)

provide more challenging opponents. It also protects against the easy exploitability and fragility of hard-coded opponent strategies. Spronck et al (2006) demonstrate the idea by an implementation of dynamic scripting for tactical combat in the RPG *Neverwinter Nights*. The AI trained by dynamic scripting learned to overcome fixed AIs after 20-30 episodes.

Very few video games have in-game adaptation, with notable exceptions *Black & White* and *Creatures* and the academic game *NERO* (Stanley et al, 2005). The three games have lots in common: learning is central part of the gameplay, and the player can observe how his creatures are interacting with the environment. The player can train his creatures by rewarding/punishing them and modifying their environment (for example, to set up tasks or provide new stimuli).

There are two main reasons why so few examples of in-game learning exist: learning speed and reliability. To have any noticeable effect, learning should be visible after a few repetitions (or few dozens at most). Spronck et al (2006)'s application of dynamic scripting is on the verge of usability in this respect. As for reliability, learning is affected by many factors including randomness. Thus, it is unpredictable how the learning process will end, and results will probably have significant variance and a chance of failure. This kind of unpredictability is not acceptable in commercial games, with the possible exception of "teaching games" like *Black & White*, where the player controls the learning process.

17.4.2 Development-Time Learning

Opportunities are wider for offline adaptation, when all learning is done before the release of the game, and the resulting policies are then tested and fixed by the developers. For example, opponents were trained by AI techniques in the racing games *Re-Volt*, *Colin McRae Rally* and *Forza Motorsport*. We highlight two areas where offline learning has promising results: micromanagement and game balancing.

17.4.2.1 Micromanagement and Counsellors

A major problem with real-time strategy games and turn-based strategy games like the *Civilization* series is the need for micromanagement (Wender and Watson, 2009): in parallel to making high level decisions (where to attack, what kind of technology to research), the player needs to make lower-level decisions, like assigning production and allocate workforce in each city and manage individual soldiers and workers. Micromanagement tasks are tedious and repetitive. In turn-based strategy games, a standard solution to avoid micromanagement is to hire AI counsellors who can take part of the job; in RTS games, counsellors are not yet common. Szczepański and Aamodt (2009) apply case-based learning to micromanagement in the RTS *Warcraft 3*. According to their experiments, the agents (that, for example, learned to heal wounded units in battle), were favorably received by beginner and

intermediate players (though not by experts who were good in micromanagement and were confused by overridden commands). Sharifi et al (2010) train companion behaviors in the RPG *Neverwinter Nights* using Sarsa(λ). The companion gets rewards from two sources: an ‘internal’ reward for successful/failed completion of a task (e.g., punishment for failed attempts to disarm a trap), and optional, sparse ‘verbal rewards’ received from the player.

17.4.2.2 Game Balancing

Game balance can refer to two things:

- The balanced winning chances of opponents in games with asymmetrical design. Asymmetry can be light, e.g., in an RTS game where different races start with different strengths and weaknesses, or stronger, like in the board game *Last Night on Earth* where one player controls weak but plentiful zombies, while the other controls a small group of (quite powerful) survivors.
- The balance between the strategies of a single player.

The first kind of balance is clearly necessary for fairness, while the second kind is important to ensure that the player is not forced to repeat a single dominant strategy every time but there is truly a greater variety of options so that he can make meaningful decisions and possibly have more fun. Neither kinds of balance are easy to establish, requiring extensive playtesting—and imbalances may still go undetected during testing. This happened, for example, with tank rush in RTS *Command and Conquer*, or the collectible card game *Magic: The Gathering*, where some cards were so overpowered that they had to be restricted or banned in tournaments. Kalles and Kanellopoulos (2001) used TD-learning to test the balancedness of an abstract board game.

17.5 Closing Remarks

Games are a thriving area for reinforcement learning research, with a large number of applications. Temporal-difference learning, Monte-Carlo tree search and evolutionary reinforcement learning are among the most popular techniques applied. In many cases, RL approaches are competitive with other AI techniques and/or human intelligence, while in others this did not happen yet, but improvement is rapid.

The chapter tried to give a representative sample of problems and challenges specific to games, and the major approaches used in game applications. In the review we focused on understanding the tips and tricks that make the difference between a working and failing RL algorithm. By no means do we claim to have a complete overview of the RL techniques used in games. We did not even try to give a historical overview; and we had to leave out most results bordering evolutionary computation, game theory, supervised learning and general AI research. Furthermore, as the focus of the chapter was the RL side of the RL–games relationship, we had to omit whole genres of games, like turn-based strategy games,

role-playing games, or (from a more theoretical perspective) iterated matrix games. Luckily, there are many high-quality review papers covering these topics. For classical board and card games, Fürnkranz (2001); Fürnkranz (2007) focuses on machine learning techniques, Schaeffer (2000); van den Herik et al (2002) and the recent book of Mańdziuk (2010) surveys general AI approaches including machine learning (and within that, RL). Ghory (2004) gives a good overview of TD variants applied to classical board games; while Galway et al (2008) focuses on evolutionary methods and computer games.

Probably the most important lesson to be learned is: RL techniques are powerful, but to make them work, the supporting components are just as important, including a proper representation, training schedule, game-specific domain knowledge and heuristics. While these things do not suffice in themselves to create strong game-playing agents, neither do basic reinforcement learning methods in their vanilla form – even TD-Gammon needed the support of other components (Tesauro, 2002).

Starting research in RL and games has low entry costs. Many games are easily accessible, together with some supporting AI components. Most classical board and card games have strong open-source engines, and open-source clones also exist for many computer games. Emulators for older game systems like the Atari 2600 give access to hundreds of the older generation of video games. Furthermore, many of the state-of-the-art games give access to their AI system either through API or through scripting languages, allowing relatively painless implementation of RL agents. Finally, multiple competitions offer venues for comparing AI implementations to computer games, for example, the Pac-Man competitions organized annually on the IEEE CIG conference,²² the Mario AI Championship,²³ or the Tetris, Mario and RTS domains of earlier RL-Competitions.²⁴

Reinforcement learning is a promising area of research with lots of challenges and open problems left to solve. Games are exciting, and many of them are still not conquered by AI. The opportunities are many for everyone.

Acknowledgements. The author gratefully acknowledges the help of Csaba Szepesvári, Mike Bowling, Thomas Degris, Gábor Balázs, Joseph Modayil, and Hengshuai Yao. Discussions with them, their comments and suggestions helped greatly in improving the chapter, just like the comments of the anonymous reviewers.

References

- Aha, D.W., Molineaux, M., Ponsen, M.: Learning to win: Case-based plan selection in a real-time strategy game. *Case-Based Reasoning Research and Development*, 5–20 (2005)
Amit, A., Markovitch, S.: Learning to bid in bridge. *Machine Learning* 63(3), 287–327 (2006)

²² <http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html>

²³ <http://www.marioai.org/>

²⁴ <http://rl-competition.org/>

- Andrade, G., Santana, H., Furtado, A., Leitão, A., Ramalho, G.: Online adaptation of computer games agents: A reinforcement learning approach. *Scientia* 15(2) (2004)
- Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47, 235–256 (2002)
- Bartók, G., Szepesvári, C., Zilles, S.: Models of active learning in group-structured state spaces. *Information and Computation* 208, 364–384 (2010)
- Baxter, J., Tridgell, A., Weaver, L.: Learning to play chess using temporal-differences. *Machine learning* 40(3), 243–263 (2000)
- Baxter, J., Tridgell, A., Weaver, L.: Reinforcement learning and chess. In: *Machines that learn to play games*, pp. 91–116. Nova Science Publishers, Inc. (2001)
- Beal, D., Smith, M.C.: Learning piece values using temporal differences. *ICCA Journal* 20(3), 147–151 (1997)
- Bertsekas, D.P., Tsitsiklis, J.N.: *Neuro-Dynamic Programming*. Athena Scientific (1996)
- Billings, D., Davidson, A., Schauenberg, T., Burch, N., Bowling, M., Holte, R.C., Schaeffer, J., Szafron, D.: Game-Tree Search with Adaptation in Stochastic Imperfect-Information Games. In: van den Herik, H.J., Björnsson, Y., Netanyahu, N.S. (eds.) CG 2004. LNCS, vol. 3846, pp. 21–34. Springer, Heidelberg (2006)
- Björnsson, Y., Finnsson, H.: Cadiaplayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games* 1(1), 4–15 (2009)
- Böhm, N., Kókai, G., Mandl, S.: Evolving a heuristic function for the game of tetris. In: Proc. Lernen, Wissensentdeckung und Adaptivität LWA, pp. 118–122 (2004)
- Boumaza, A.: On the evolution of artificial Tetris players. In: *IEEE Symposium on Computational Intelligence and Games* (2009)
- Bouzy, B., Helmstetter, B.: Monte Carlo Go developments. In: *Advances in Computer Games*, pp. 159–174 (2003)
- Bowling, M.: Convergence and no-regret in multiagent learning. In: *Neural Information Processing Systems*, pp. 209–216 (2004)
- Buro, M.: From simple features to sophisticated evaluation functions. In: *International Conference on Computers and Games*, pp. 126–145 (1998)
- Buro, M., Furtak, T.: RTS games as test-bed for real-time research. *JCIS*, 481–484 (2003)
- Buro, M., Lanctot, M., Orsten, S.: The second annual real-time strategy game AI competition. In: *GAME-ON NA* (2007)
- Chaslot, G., Winands, M., Herik, H., Uiterwijk, J., Bouzy, B.: Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation* 4(3), 343 (2008)
- Chaslot, G., Fiter, C., Hoock, J.B., Rimmel, A., Teytaud, O.: Adding Expert Knowledge and Exploration in Monte-Carlo Tree Search. In: van den Herik, H.J., Spronck, P. (eds.) ACG 2009. LNCS, vol. 6048, pp. 1–13. Springer, Heidelberg (2010)
- Chariot, L., Gelly, S., Jean-Baptiste, H., Perez, J., Rimmel, A., Teytaud, O.: Including expert knowledge in bandit-based Monte-Carlo planning, with application to computer-Go. In: European Workshop on Reinforcement Learning (2008)
- Coquelin, P.A., Munos, R.: Bandit algorithms for tree search. In: *Uncertainty in Artificial Intelligence* (2007)
- Coulom, R.: Efficient Selectivity and Backup Operators in Monte-carlo Tree Search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M(J.) (eds.) CG 2006. LNCS, vol. 4630, pp. 72–83. Springer, Heidelberg (2007)
- Coulom, R.: Computing Elo ratings of move patterns in the game of go. *ICGA Journal* 30(4), 198–208 (2007)
- Dahl, F.A.: Honte, a Go-playing program using neural nets. In: *Machines that learn to play games*, pp. 205–223. Nova Science Publishers (2001)

- Davidson, A.: Opponent modeling in poker: Learning and acting in a hostile and uncertain environment. Master's thesis, University of Alberta (2002)
- Diuk, C., Cohen, A., Littman, M.L.: An object-oriented representation for efficient reinforcement learning. In: International Conference on Machine Learning, pp. 240–247 (2008)
- Droste, S., Fürnkranz, J.: Learning of piece values for chess variants. Tech. Rep. TUD-KE-2008-07, Knowledge Engineering Group, TU Darmstadt (2008)
- Džeroski, S., Raedt, L.D., Driessens, K.: Relational reinforcement learning. *Machine Learning* 43(1-2), 7–52 (2001)
- Epstein, S.L.: Toward an ideal trainer. *Machine Learning* 15, 251–277 (1994)
- Farias, V.F., van Roy, B.: Tetris: A Study of Randomized Constraint Sampling. In: Probabilistic and Randomized Methods for Design Under Uncertainty. Springer, UK (2006)
- Fawcett, T., Utgoff, P.: Automatic feature generation for problem solving systems. In: International Conference on Machine Learning, pp. 144–153 (1992)
- Finkelstein, L., Markovitch, S.: Learning to play chess selectively by acquiring move patterns. *ICCA Journal* 21, 100–119 (1998)
- Fudenberg, D., Levine, D.K.: The theory of learning in games. MIT Press (1998)
- Fürnkranz, J.: Machine learning in games: a survey. In: *Machines that Learn to Play Games*, pp. 11–59. Nova Science Publishers (2001)
- Fürnkranz, J.: Recent advances in machine learning and game playing. Tech. rep., TU Darmstadt (2007)
- Galway, L., Charles, D., Black, M.: Machine learning in digital games: a survey. *Artificial Intelligence Review* 29(2), 123–161 (2008)
- Gelly, S., Silver, D.: Achieving master-level play in 9x9 computer go. In: AAAI, pp. 1537–1540 (2008)
- Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with patterns in Monte-Carlo go. Tech. rep., INRIA (2006)
- Gherrity, M.: A game-learning machine. PhD thesis, University of California, San Diego, CA (1993)
- Ghory, I.: Reinforcement learning in board games. Tech. rep., Department of Computer Science, University of Bristol (2004)
- Gilgenbach, M.: Fun game AI design for beginners. In: *AI Game Programming Wisdom*, vol. 3. Charles River Media, Inc. (2006)
- Gilpin, A., Sandholm, T.: Lossless abstraction of imperfect information games. *Journal of the ACM* 54(5), 25 (2007)
- Gilpin, A., Sandholm, T., Sørensen, T.B.: Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In: AAAI, vol. 22, pp. 50–57 (2007)
- Ginsberg, M.L.: Gib: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research* 14, 313–368 (2002)
- Gould, J., Levinson, R.: Experience-based adaptive search. Tech. Rep. UCSC-CRL-92-10, University of California at Santa Cruz (1992)
- Günther, M.: Automatic feature construction for general game playing. PhD thesis, Dresden University of Technology (2008)
- Hagelbäck, J., Johansson, S.J.: Measuring player experience on runtime dynamic difficulty scaling in an RTS game. In: International Conference on Computational Intelligence and Games (2009)
- Hartley, T., Mehdi, Q., Gough, N.: Online learning from observation for interactive computer games. In: International Conference on Computer Games: Artificial Intelligence and Mobile Systems, pp. 27–30 (2005)

- van den Herik, H.J., Uiterwijk, J.W.H.M., van Rijswijck, J.: Games solved: Now and in the future. *Artificial Intelligence* 134, 277–311 (2002)
- Hsu, F.H.: Behind Deep Blue: Building the Computer that Defeated the World Chess Champion. Princeton University Press, Princeton (2002)
- Hunicke, R., Chapman, V.: AI for dynamic difficult adjustment in games. In: Challenges in Game AI Workshop (2004)
- Kakade, S.: A natural policy gradient. In: Advances in Neural Information Processing Systems, vol. 14, pp. 1531–1538 (2001)
- Kalles, D., Kanellopoulos, P.: On verifying game designs and playing strategies using reinforcement learning. In: ACM Symposium on Applied Computing, pp. 6–11 (2001)
- Kerbusch, P.: Learning unit values in Wargus using temporal differences. BSc thesis (2005)
- Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
- Kocsis, L., Szepesvári, C., Winands, M.H.M.: RSPSA: Enhanced Parameter Optimization in Games. In: van den Herik, H.J., Hsu, S.-C., Hsu, T.-S., Donkers, H.H.L.M.(J.) (eds.) CG 2005. LNCS, vol. 4250, pp. 39–56. Springer, Heidelberg (2006)
- Kok, E.: Adaptive reinforcement learning agents in RTS games. Master's thesis, University of Utrecht, The Netherlands (2008)
- Koza, J.: Genetic programming: on the programming of computers by means of natural selection. MIT Press (1992)
- Kuhlmann, G.J.: Automated domain analysis and transfer learning in general game playing. PhD thesis, University of Texas at Austin (2010)
- Lagoudakis, M.G., Parr, R., Littman, M.L.: Least-Squares Methods in Reinforcement Learning for Control. In: Vlahavas, I.P., Spyropoulos, C.D. (eds.) SETN 2002. LNCS (LNAI), vol. 2308, pp. 249–260. Springer, Heidelberg (2002)
- Laursen, R., Nielsen, D.: Investigating small scale combat situations in real time strategy computer games. Master's thesis, University of Aarhus (2005)
- Levinson, R., Weber, R.: Chess Neighborhoods, Function Combination, and Reinforcement Learning. In: Marsland, T., Frank, I. (eds.) CG 2001. LNCS, vol. 2063, pp. 133–150. Springer, Heidelberg (2002)
- Lorenz, U.: Beyond Optimal Play in Two-Person-Zerosum Games. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 749–759. Springer, Heidelberg (2004)
- Mańdziuk, J.: Knowledge-Free and Learning-Based Methods in Intelligent Game Playing. Springer, Heidelberg (2010)
- Marthi, B., Russell, S., Latham, D.: Writing Stratagus-playing agents in concurrent alisp. In: IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games, pp. 67–71 (2005)
- McGlinchey, S.J.: Learning of AI players from game observation data. In: GAME-ON, pp. 106–110 (2003)
- Molineaux, M., Aha, D.W., Ponsen, M.: Defeating novel opponents in a real-time strategy game. In: IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games, pp. 72–77 (2005)
- Moriarty, D.E., Miikkulainen, R.: Discovering complex Othello strategies through evolutionary neural networks. *Connection Science* 7, 195–209 (1995)
- Müller, M.: Position evaluation in computer go. *ICGA Journal* 25(4), 219–228 (2002)
- Naddaf, Y.: Game-independent AI agents for playing Atari 2600 console games. Master's thesis, University of Alberta (2010)

- Pollack, J.B., Blair, A.D.: Why did TD-Gammon work? In: Neural Information Processing Systems, vol. 9, pp. 10–16 (1997)
- Ponsen, M., Spronck, P.: Improving adaptive game AI with evolutionary learning. In: Computer Games: Artificial Intelligence, Design and Education (2004)
- Ponsen, M., Muñoz-Avila, H., Spronck, P., Aha, D.W.: Automatically acquiring adaptive real-time strategy game opponents using evolutionary learning. In: Proceedings of the 17th Innovative Applications of Artificial Intelligence Conference (2005)
- Ponsen, M., Spronck, P., Tuyls, K.: Hierarchical reinforcement learning in computer games. In: Adaptive Learning Agents and Multi-Agent Systems, pp. 49–60 (2006)
- Ponsen, M., Taylor, M.E., Tuyls, K.: Abstraction and Generalization in Reinforcement Learning: A Summary and Framework. In: Taylor, M.E., Tuyls, K. (eds.) ALA 2009. LNCS, vol. 5924, pp. 1–33. Springer, Heidelberg (2010)
- Ramanujan, R., Sabharwal, A., Selman, B.: Adversarial search spaces and sampling-based planning. In: International Conference on Automated Planning and Scheduling (2010)
- Risk, N., Szafron, D.: Using counterfactual regret minimization to create competitive multi-player poker agents. In: International Conference on Autonomous Agents and Multiagent Systems, pp. 159–166 (2010)
- Rubin, J., Watson, I.: Computer poker: A review. *Artificial Intelligence* 175(5-6), 958–987 (2011)
- Schaeffer, J.: The games computers (and people) play. In: Zelkowitz, M. (ed.) Advances in Computers, vol. 50, pp. 89–266. Academic Press (2000)
- Schaeffer, J., Hlynka, M., Jussila, V.: Temporal difference learning applied to a high-performance game-playing program. In: International Joint Conference on Artificial Intelligence, pp. 529–534 (2001)
- Schnizlein, D., Bowling, M., Szafron, D.: Probabilistic state translation in extensive games with large action sets. In: International Joint Conference on Artificial Intelligence, pp. 278–284 (2009)
- Schraudolph, N.N., Dayan, P., Sejnowski, T.J.: Learning to evaluate go positions via temporal difference methods. In: Computational Intelligence in Games. Studies in Fuzziness and Soft Computing, ch. 4, vol. 62, pp. 77–98. Springer, Heidelberg (2001)
- Scott, B.: The illusion of intelligence. In: AI Game Programming Wisdom, pp. 16–20. Charles River Media (2002)
- Shapiro, A., Fuchs, G., Levinson, R.: Learning a Game Strategy Using Pattern-Weights and Self-Play. In: Schaeffer, J., Müller, M., Björnsson, Y. (eds.) CG 2002. LNCS, vol. 2883, pp. 42–60. Springer, Heidelberg (2003)
- Sharifi, A.A., Zhao, R., Szafron, D.: Learning companion behaviors using reinforcement learning in games. In: AIIDE (2010)
- Sharma, S., Kobti, Z., Goodwin, S.: General game playing: An overview and open problems. In: International Conference on Computing, Engineering and Information, pp. 257–260 (2009)
- Silver, D., Tesauro, G.: Monte-carlo simulation balancing. In: International Conference on Machine Learning (2009)
- Silver, D., Sutton, R., Mueller, M.: Sample-based learning and search with permanent and transient memories. In: ICML (2008)
- Spronck, P., Sprinkhuizen-Kuyper, I., Postma, E.: Difficulty scaling of game AI. In: GAME-ON 2004: 5th International Conference on Intelligent Games and Simulation (2004)
- Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., Postma, E.: Adaptive game AI with dynamic scripting. *Machine Learning* 63(3), 217–248 (2006)

- Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation* 9(6), 653–668 (2005)
- Sturtevant, N., White, A.: Feature construction for reinforcement learning in Hearts. In: *Advances in Computers and Games*, pp. 122–134 (2007)
- Szczepański, T., Aamodt, A.: Case-based reasoning for improved micromanagement in real-time strategy games. In: *Workshop on Case-Based Reasoning for Computer Games, 8th International Conference on Case-Based Reasoning*, pp. 139–148 (2009)
- Szita, I., Lörincz, A.: Learning Tetris using the noisy cross-entropy method. *Neural Computation* 18(10), 2936–2941 (2006a)
- Szita, I., Lörincz, A.: Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man. *Journal of Artificial Intelligence Research* 30, 659–684 (2006b)
- Szita, I., Szepesvári, C.: Sz-tetris as a benchmark for studying key problems of rl. In: *ICML 2010 Workshop on Machine Learning and Games* (2010)
- Szita, I., Chaslot, G., Spronck, P.: Monte-Carlo Tree Search in Settlers of Catan. In: van den Herik, H.J., Spronck, P. (eds.) *ACG 2009. LNCS*, vol. 6048, pp. 21–32. Springer, Heidelberg (2010)
- Tesauro, G.: Practical issues in temporal difference learning. *Machine Learning* 8, 257–277 (1992)
- Tesauro, G.: Temporal difference learning and TD-gammon. *Communications of the ACM* 38(3), 58–68 (1995)
- Tesauro, G.: Comments on co-evolution in the successful learning of backgammon strategy'. *Machine Learning* 32(3), 241–243 (1998)
- Tesauro, G.: Programming backgammon using self-teaching neural nets. *Artificial Intelligence* 134(1-2), 181–199 (2002)
- Thiery, C., Scherrer, B.: Building controllers for Tetris. *ICGA Journal* 32(1), 3–11 (2009)
- Thrun, S.: Learning to play the game of chess. In: *Neural Information Processing Systems*, vol. 7, pp. 1069–1076 (1995)
- Utgoff, P.: Feature construction for game playing. In: Fürnkranz, J., Kubat, M. (eds.) *Machines that Learn to Play Games*, pp. 131–152. Nova Science Publishers (2001)
- Utgoff, P., Precup, D.: Constructive function approximation. In: Liu, H., Motoda, H. (eds.) *Feature Extraction, Construction and Selection: A Data Mining Perspective*, vol. 453, pp. 219–235. Kluwer Academic Publishers (1998)
- Veness, J., Silver, D., Uther, W., Blair, A.: Bootstrapping from game tree search. In: *Neural Information Processing Systems*, vol. 22, pp. 1937–1945 (2009)
- Weber, B.G., Mateas, M.: Case-based reasoning for build order in real-time strategy games. In: *Artificial Intelligence and Interactive Digital Entertainment*, pp. 1313–1318 (2009)
- Wender, S., Watson, I.: Using reinforcement learning for city site selection in the turn-based strategy game Civilization IV. In: *Computational Intelligence and Games*, pp. 372–377 (2009)
- Wiering, M.A.: Self-play and using an expert to learn to play backgammon with temporal difference learning. *Journal of Intelligent Learning Systems and Applications* 2, 57–68 (2010)
- Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret minimization in games with incomplete information. In: *Neural Information Processing Systems*, pp. 1729–1736 (2008)

Chapter 18

Reinforcement Learning in Robotics: A Survey

Jens Kober and Jan Peters

Abstract. As most action generation problems of autonomous robots can be phrased in terms of sequential decision problems, robotics offers a tremendously important and interesting application platform for reinforcement learning. Similarly, the real-world challenges of this domain pose a major real-world check for reinforcement learning. Hence, the interplay between both disciplines can be seen as promising as the one between physics and mathematics. Nevertheless, only a fraction of the scientists working on reinforcement learning are sufficiently tied to robotics to oversee most problems encountered in this context. Thus, we will bring the most important challenges faced by robot reinforcement learning to their attention. To achieve this goal, we will attempt to survey most work that has successfully applied reinforcement learning to behavior generation for real robots. We discuss how the presented successful approaches have been made tractable despite the complexity of the domain and will study how representations or the inclusion of prior knowledge can make a significant difference. As a result, a particular focus of our chapter lies on the choice between model-based and model-free as well as between value function-based and policy search methods. As a result, we obtain a fairly complete survey of robot reinforcement learning which should allow a general reinforcement learning researcher to understand this domain.

18.1 Introduction

Robotics has a near infinite amount of interesting learning problems, a large percentage of which can be phrased as reinforcement learning problems. See Figure 18.1 for

Jens Kober

Intelligent Autonomous Systems Institute, Technische Universitaet Darmstadt, Darmstadt, Germany

Jan Peters

Robot Learning Lab, Max-Planck Institute for Intelligent Systems, Tübingen, Germany
e-mail: {jens.kober, jan.peters}@tuebingen.mpg.de

an illustration of the wide variety of robots that have learned tasks using reinforcement learning. However, robotics as a domain differs significantly from well-defined typical reinforcement learning benchmark problems, which usually have discrete states and actions. In contrast, many real-world problems in robotics are best represented with high-dimensional, continuous states and actions. Every single trial run is costly and, as a result, such applications force us to focus on problems that do not arise that frequently in classical reinforcement learning benchmark examples. In this book chapter, we highlight the challenges faced in robot reinforcement learning and bring many of the inherent problems of this domain to the reader's attention.

Robotics is characterized by high dimensionality due to the many degrees of freedom of modern anthropomorphic robots. Experience on the real system is costly and often hard to reproduce. However, it usually cannot be replaced by simulations, at least for highly dynamic tasks, as even small modeling errors accumulate to substantially different dynamic behavior. Another challenge faced in robot reinforcement learning is the generation of appropriate reward functions. Good rewards that lead the systems quickly to success are needed to cope with the cost of real-world experience but are a substantial manual contribution.

Obviously, not every reinforcement learning method is equally suitable for the robotics domain. In fact, many of the methods that scale to more interesting tasks are model-based (Atkeson et al, 1997; Abbeel et al, 2007) and often employ policy search rather than value function-based approaches (Gullapalli et al, 1994; Miyamoto et al, 1996; Kohl and Stone, 2004; Tedrake et al, 2005; Peters and Schaal, 2008a,b; Kober and Peters, 2009). This stands in contrast to much of mainstream reinforcement (Kaelbling et al, 1996; Sutton and Barto, 1998). We attempt to give a fairly complete overview on real robot reinforcement learning citing most original papers while distinguishing mainly on a methodological level.

As none of the presented methods extends to robotics with ease, we discuss how robot reinforcement learning can be made tractable. We present several approaches to this problem such as choosing an appropriate representation for your value function or policy, incorporating prior knowledge, and transfer from simulations.

In this book chapter, we survey real robot reinforcement learning and highlight how these approaches were able to handle the challenges posed by this setting. Less attention is given to results that correspond only to slightly enhanced grid-worlds or that were learned exclusively in simulation. The challenges in applying reinforcement learning in robotics are discussed in Section 18.2.

Standard reinforcement learning methods suffer from the discussed challenges. As already pointed out in the reinforcement learning review paper by Kaelbling et al (1996) “we must give up tabula rasa learning techniques and begin to incorporate bias that will give leverage to the learning process”. Hence, we concisely present reinforcement learning techniques in the context of robotics in Section 18.3. Different approaches of making reinforcement learning tractable are treated in Sections 18.4 to 18.6. Finally in Section 18.7, we employ the example of ball-in-a-cup to highlight which of the various approaches discussed in the book chapter have been particularly helpful for us to make such a complex task tractable. In Section 18.8, we give a conclusion and outlook on interesting problems.

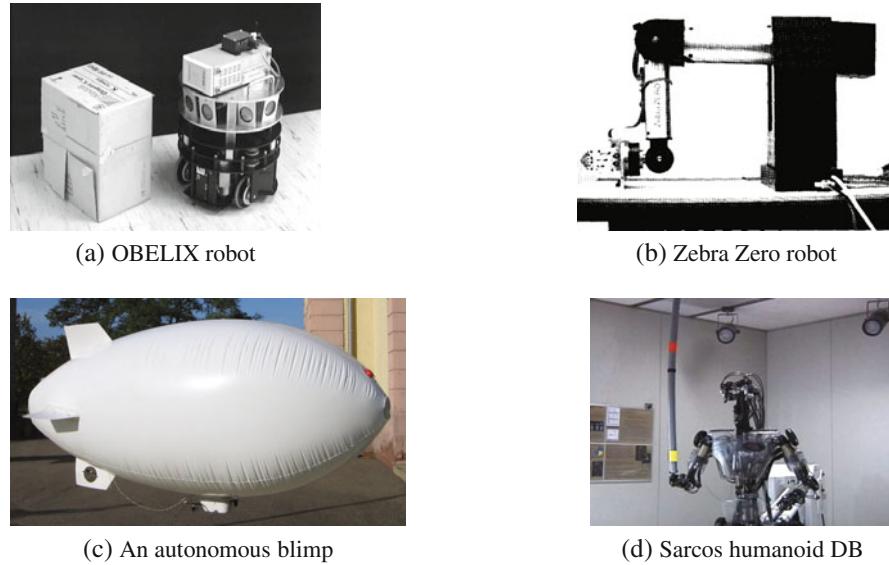


Fig. 18.1 This figure illustrates robots to which reinforcement learning has been applied. The robots cover the whole range of wheeled mobile robots, robotic arms, autonomous vehicles, to humanoid robots. (a) The OBELIX robot is a wheeled mobile robot that learned to push boxes (Mahadevan and Connell, 1992) with a value function-based approach (Picture reprint with permission of Sridhar Mahadevan). (b) The Zebra Zero robot is a robot arm that learned a peg-in-hole insertion task (Gullapalli et al, 1994) with a model-free policy gradient approach (Picture reprint with permission of Rod Grupen). (c) The control of such autonomous blimps (Picture reprint with permission of Axel Rottmann) was learned with both a value function based approach (Rottmann et al, 2007) and model-based policy search (Ko et al, 2007). (d) The Sarcos humanoid DB learned a pole-balancing task (Schaal, 1997) using forward models (Picture reprint with permission of Stefan Schaal).

18.2 Challenges in Robot Reinforcement Learning

Reinforcement learning is generically a hard problem and many of its challenges apply particularly in the robotics setting. As the states and actions of most robots are inherently continuous (see Chapter 7) and the dimensionality can be high, we face the ‘Curse of Dimensionality’ (Bellman, 1957) as discussed in Section 18.2.1. As we deal with complex physical systems, samples can be expensive due to the long execution time of complete tasks, required manual interventions as well as maintenance and repair (see Section 18.2.2). A robot requires that the algorithm runs in real-time and that it is capable of dealing with delays in the sensing and execution which are inherent in physical systems (see Section 18.2.3). Obviously, a simulation could alleviate many problems but direct transfer from a simulated

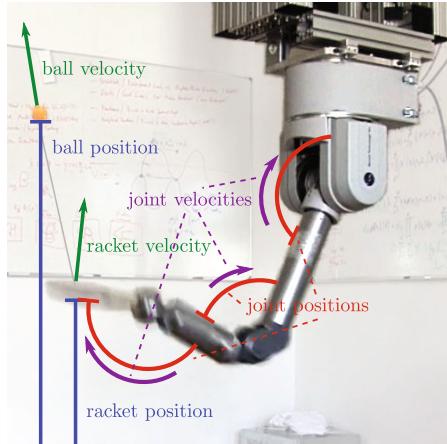


Fig. 18.2 This Figure illustrates the state space of a robot reinforcement learning task

to a real robot is challenging due to model errors as discussed in Section 18.2.4. An often underestimated problem is the goal specification, which is achieved by designing a good reward function. As noted in Section 18.2.5 this choice can make the difference between feasibility and an unreasonable amount of exploration.

18.2.1 Curse of Dimensionality

When Bellman (1957) explored optimal control in continuous states spaces, he faced an exponential explosion of discrete states and actions for which he coined the term ‘Curse of Dimensionality’. Robotic systems inherently have to deal with such continuous states and actions. For example, in a ball-paddling task as shown in Figure 18.2, a proper representation of a robot’s state would consist of its joint angles and velocities for each of its seven degrees of freedom as well as Cartesian position and velocity of the ball. The robot’s actions would be the generated motor commands which often are torques or accelerations. In this example, we have a $2 \times (7 + 3) = 20$ state dimensions and 7-dimensional continuous actions. Obviously, other tasks may require even more dimensions, e.g., human-like actuation often follows the antagonistic principle which additionally allows to control stiffness. Obviously such dimensionality is a major challenge for both the robotics and the reinforcement learning communities.

In robotics, tasks with such problems are often made more accessible to the robot engineer by shifting some of the complexity to a lower layer of functionality. In the ball paddling example, we can simplify by controlling the robot in racket space (which is lower-dimensional as the racket is orientation-invariant around the string’s mounting point) with an operational space control law (Nakanishi et al, 2008). Many

commercial robot systems also encapsulate some of the state and action components in an embedded control system (e.g., trajectory fragments are frequently used as actions for industrial robots); however, this form of a state dimensionality reduction severely limits the dynamic capabilities of the robot according to our experience (Schaal et al, 2002; Peters et al, 2010b).

The reinforcement learning community has a long history of dealing with dimensionality using computational abstractions. It offers a larger set of applicable tools ranging from adaptive discretizations (Buşoniu et al, 2010) over function approximation approaches (Sutton and Barto, 1998) to macro actions or options (Barto and Mahadevan, 2003). Macro actions would allow decomposing a task in elementary components and quite naturally translate to robotics. For example, a macro action “move one meter to the left” could be achieved by a lower level controller that takes care of accelerating, moving, and stopping while ensuring the precision. Using a limited set of manually generated macro actions, standard reinforcement learning approaches can be made tractable for navigational tasks for mobile robots. However, the automatic generation of such sets of macro actions is the key issue in order to enable such approaches. We will discuss approaches that have been successful in robot reinforcement learning in Section 18.4.

18.2.2 Curse of Real-World Samples

Robots inherently interact with the real world and, hence, robot reinforcement learning suffers from most of the resulting real world problems. For example, robot hardware is usually expensive, suffers from wear and tear, and requires careful maintenance. Repairing a robot system is a non-negligible effort associated with cost, physical labor and long waiting periods. Hence, to apply reinforcement learning in robotics, safe exploration becomes a key issue of the learning process; a problem often neglected in the general reinforcement learning community.

However, several more aspects of the real world make robotics a challenging domain. As the dynamics of a robot can change due to many external factors ranging from temperature to wear, the learning process may never fully converge, i.e., it needs a ‘tracking solution’ (Sutton et al, 2007). Frequently, the environment settings during an earlier learning period cannot be reproduced and the external factors are not clear, e.g., how did the light conditions affect the performance of the vision system and, as a result, the task’s performance. This problem makes comparisons of algorithms particularly hard.

Most real robot learning tasks require some form of human supervision, e.g., putting the pole back on the robot’s end-effector during pole balancing, see Figure 18.1d, after a failure. Even when an automatic reset exists (e.g., by having a smart contraption that resets the pole), learning speed becomes essential as a task on a real robot cannot be sped up. The whole episode needs to be complete as it is often not possible to start from arbitrary states.

For such reasons, real-world samples are expensive in terms of time, labor and, potentially, finances. Thus, sample efficient algorithms that are able to learn from a small number of trials are essential. In Sections 18.6.2 and 18.6.3 we will discuss several approaches that allow reducing the amount of required real world interactions.

18.2.3 Curse of Real-World Interactions

As the robot is a physical system there are strict constraints on the interaction between the learning algorithm and the robot setup. Usually the robot needs to get commands at fixed frequency and for dynamic tasks the movement cannot be paused. Thus, the agent has to select actions in real-time. It is often not possible to pause to think, learn or plan between each action but rather the learning algorithm has to deal with a fixed amount of time. Thus, not only are samples expensive to obtain, but also often only a very limited number of samples can be used, if the runtime of the algorithms depends on the number of samples. These constraints are less severe in an episodic setting where the time intensive part of the learning can be postponed to the period between episodes.

On physical systems there are always delays in sensing and actuation. The state of the setup represented by the sensors slightly lags behind the real state due to processing and communication delays. More critically there are also communication delays in the actuation as well as delays due to the fact that a physical cannot instantly change its movement. For example, due to inertia the direction of a movement cannot be inverted but rather a breaking and accelerating phase are required. The robot always needs this kind of a transition phase. Due to these delays, actions do not have instantaneous effects but are observable only several time steps later. In contrast, in most general reinforcement learning algorithms, the actions are assumed to take effect instantaneously.

18.2.4 Curse of Model Errors

One way to offset the cost of real world interaction would be accurate models that are being used as simulators. In an ideal setting, such an approach would render it possible to learn the behavior in simulation and subsequently transfer it to the real robot. Unfortunately, creating a sufficiently accurate model of the robot and the environment is challenging. As small model errors may accumulate, we can frequently see a fast divergence of the simulated robot from the real-world system. When a policy is trained using an imprecise forward model as simulator, the behavior will not transfer without significant modifications as experienced by Atkeson (1994) when learning the underactuated swing-up. Only in a limited number of experiments, the authors have achieved such a direct transfer, see Section 18.6.3 for examples. If the task is energy absorbing or excessive energy is not critical, it is safer to assume that

approaches that can be applied in simulation may work similarly in the real-world (Kober and Peters, 2010). In an energy absorbing scenario, the task is inherently stable and transferring policies poses a low risk of damaging the robot. However, in energy absorbing scenarios, tasks can often be learned better in the real world than in simulation due to complex interactions between mechanical objects. In an energy emitting scenario, the task is inherently unstable and transferring policies poses a high risk. As we will see later, models are best used to test the algorithms in simulations but can also be used to check the proximity to theoretically optimal solutions, or to perform ‘mental rehearsal’.

18.2.5 Curse of Goal Specification

In reinforcement learning, the goal of the task is implicitly specified by the reward. Defining a good reward function in robot reinforcement learning is hence often a daunting task. Giving rewards only upon task achievement, e.g., did a table tennis robot win the match, will result in a simple binary reward. However, the robot would receive such a reward so rarely that it is unlikely to ever succeed in the lifetime of a real-world system. Hence, instead of using only simpler binary rewards, we frequently need to include additional knowledge into such scalar rewards to guide the learning process to a reasonable solution. The trade-off between different factors may be essential as hitting a table tennis ball very fast will result in a high score but is likely to destroy the robot. Good reinforcement learning algorithms often exploit the reward function in unexpected ways, especially if the reinforcement learning is done locally and not globally. For example, if contact between racket and ball is part of the reward in ball paddling (see Figure 18.2), many locally optimal solutions would attempt to simply keep the ball on the racket.

Inverse reinforcement learning, also known as apprenticeship learning, is a promising alternative to specifying the reward function manually. Instead, it assumes that the reward function can be reconstructed from a set of expert demonstrations. Recently, it has yielded a variety of successful applications in robotics, see (Kolter et al, 2007; Abbeel et al, 2008; Coates et al, 2009) for more information.

18.3 Foundations of Robot Reinforcement Learning

Real-world domains such as robotics are affected more strongly by the basic approach choices. Hence, we introduce reinforcement learning in this chapter with a particular point of view. As stated in Chapter 1, the goal of reinforcement learning is to find a policy $\pi(s,a)$ that gathers maximal rewards $R(s,a)$. However, in real-world domains the average reward is often more suitable than a discounted formulation due to its stability properties (Peters et al, 2004). In order to incorporate exploration, the policy is considered a conditional probability distribution $\pi(s,a) = f(a|s,\theta)$ with parameters θ . Reinforcement learning aims at finding the

optimal policy π^* or equivalent policy parameters θ^* which maximize the average return $J(\pi) = \sum_{s,a} \mu^\pi(s) \pi(s,a) R(s,a)$ where μ^π is the stationary state distribution generated by policy π acting in the environment $T(s,a,s') = P(s'|s,a)$. Hence, we have an optimization problem of

$$\max_{\mu^\pi, \pi} J(\pi) = \sum_{s,a} \mu^\pi(s) \pi(s,a) R(s,a), \quad (18.1)$$

$$\text{s.t. } \mu^\pi(s') = \sum_{s,a} \mu^\pi(s) \pi(s,a) T(s,a,s'), \forall s' \in S, \quad (18.2)$$

$$1 = \sum_{s,a} \mu^\pi(s) \pi(s,a). \quad (18.3)$$

Here, Equation (18.2) defines stationarity of the state distributions μ^π (i.e., it ensures that it converges) and Equation (18.3) ensures a proper state-action probability distribution. This optimization problem can be tackled in two substantially different ways (Bellman, 1967, 1971), i.e., we can search the optimal solution directly in the original, primal problem, and we can optimize in the dual formulation. Optimizing in primal formulation is known as *policy search* in reinforcement learning while searching in the dual is called a *value function-based approach*.

18.3.1 Value Function Approaches

Most of reinforcement has focused on solving the optimization problem in Equations (18.1-18.3) not directly but rather in its dual form. Using the Lagrangian multipliers $V(s')$ and \bar{R} , we can express the Lagrangian of the problem by

$$L = \sum_{s,a} \mu^\pi(s) \pi(s,a) \left[R(s,a) + \sum_{s'} V(s') T(s,a,s') - \bar{R} \right] - \sum_{s'} V(s') \mu^\pi(s') + \bar{R}.$$

Using straightforward insights and the stationarity condition $\sum_{s',a'} V(s') \mu^\pi(s') \pi(s',a') = \sum_{s,a} V(s) \mu^\pi(s) \pi(s,a)$, we can obtain the Karush-Kuhn-Tucker conditions (Kuhn and Tucker, 1950) by differentiating with respect to $\mu^\pi(s) \pi(s,a)$ which yields

$$\partial_{\mu^\pi} L = R(s,a) + \sum_{s'} V(s') T(s,a,s') - \bar{R} - V(s) = 0.$$

As this implies that there are as many equations the number of states multiplied by the number of actions, it is clear that only one action a^* can be optimal. Thus, we have the *Bellman Principle of Optimality* (Kirk, 1970)

$$V(s) = \max_{a^*} \left[R(s,a^*) - \bar{R} + \sum_{s'} V(s') T(s,a^*,s') \right]. \quad (18.4)$$

When evaluating Equation 18.4, we realize that $V(s)$ corresponds to the sum of the reward difference from the average reward \bar{R} encountered after taking the optimal action a^* in state s . Note that this function is usually discovered by human insight (Sutton and Barto, 1998). This principle of optimality has given birth to the field of optimal control (Kirk, 1970) and the solution above corresponds to the dynamic programming solution from the viewpoint of reinforcement learning.

Hence, we have a dual formulation of the original problem as condition for optimality. Many traditional reinforcement learning approaches are based on this equation, these are called the value function methods. Instead of directly learning a policy, they first approximate the Lagrangian multiplier $V(s)$, also called the value function, and use it to reconstruct the optimal policy. A wide variety of methods exist and can be split mainly in three classes: (i) dynamic programming-based optimal control approaches such as policy iteration or value iteration, (ii) rollout-based Monte Carlo methods and (iii) temporal difference methods such as $TD(\lambda)$, Q -learning and SARSA. See Chapter 1 for a more detailed discussion. However, such value function based approaches often do not translate well into high dimensional robotics as proper representations for the value function become intractable and even finding the optimal action can already be a hard problem. A particularly drastic problem is the error propagation in value functions where a small change in the policy may cause a large change in the value function which again causes a large change in the policy.

Table 18.1 This table illustrates different value function based reinforcement learning methods employed for robotic tasks and associated publications

VALUE FUNCTION APPROACHES	
Approach	Employed by...
<i>Model-Based</i>	Abbeel et al (2006, 2007); Atkeson and Schaal (1997); Atkeson (1998); Bagnell and Schneider (2001); Bakker et al (2006); Coates et al (2009); Donnart and Meyer (1996); Hester et al (2010); Kalmár et al (1998); Ko et al (2007); Kolter et al (2008); Martínez-Marín and Duckett (2005); Michels et al (2005); Morimoto and Doya (2001); Ng et al (2004b,a); Pendrith (1999); Schaal and Atkeson (1994); Touzet (1997); Willgoss and Iqbal (1999)
<i>Model-Free</i>	Asada et al (1996); Bakker et al (2003); Benbrahim et al (1992); Benbrahim and Franklin (1997); Birdwell and Livingston (2007); Bitzer et al (2010); Conn and Peters II (2007); Duan et al (2007, 2008); Fagg et al (1998); Gaskett et al (2000); Gräve et al (2010); Hafner and Riedmiller (2007); Huang and Weng (2002); Ilg et al (1999); Katz et al (2008); Kimura et al (2001); Kirchner (1997); Kroemer et al (2009, 2010); Latzke et al (2007); Lizotte et al (2007); Mahadevan and Connell (1992); Mataric (1997); Nemeć et al (2009, 2010); Oßwald et al (2010); Paletta et al (2007); Platt et al (2006); Riedmiller et al (2009); Rottmann et al (2007); Smart and Kaelbling (1998, 2002); Soni and Singh (2006); Thrun (1995); Tokic et al (2009); Uchibe et al (1998); Wang et al (2006)

While this may lead faster to good, possibly globally optimal solutions, such a learning process is considerably more dangerous when applied on real systems where it is likely to cause significant damage. An overview of publications using value function based methods is presented in Table 18.1. Here, model-based methods refers to all methods that employ a predetermined or a learned model.

18.3.2 Policy Search

It is straightforward to realize that the primal formulation has a lot of features relevant to robotics. It allows a natural integration of expert knowledge, e.g., through initializations of the policy. It allows domain-appropriate pre-structuring of the policy in an approximate form without changing the original problem. Optimal policies often have a lot less parameters than optimal value functions, e.g., in linear quadratic control, the value function has quadratically many parameters while the policy only requires linearly many parameters. Extensions to continuous state and action spaces follow straightforwardly. Local search in policy space can directly lead to good results as exhibited by early hill-climbing approaches (Kirk, 1970). Additional constraints can be incorporated naturally. As a result, policy search appears more natural to robotics.

Nevertheless, policy search has been considered the harder problem for a long time as the optimal solution cannot directly be determined from Equations (18.1-18.3) while the *Bellman Principle of Optimality* (Kirk, 1970) directly arises from the problems' Karush-Kuhn-Tucker conditions (Kuhn and Tucker, 1950). Notwithstanding, in robotics, policy search has recently become an important alternative to value function based methods due to the reasons described above as well as the convergence problems of approximate value function methods. Most policy search

Table 18.2 This table illustrates different policy search reinforcement learning methods employed for robotic tasks and associated publications

POLICY SEARCH	
Approach	Employed by...
<i>Gradient</i>	Deisenroth and Rasmussen (2010); Endo et al (2008); Geng et al (2006); Guenter et al (2007); Gullapalli et al (1994); Hailu and Sommer (1998); Kohl and Stone (2004); Kolter and Ng (2009); Mitsunaga et al (2005); Miyamoto et al (1996); Peters and Schaal (2008c,b); Tamei and Shibata (2009); Tedrake (2004); Tedrake et al (2005)
<i>Heuristic</i>	Erden and Leblebicioaglu (2008); Dorigo and Colombetti (1993); Mataric (1994); Svinin et al (2001); Yasuda and Ohkura (2008); Youssef (2005)
<i>Sample</i>	Buchli et al (2011); Kober and Peters (2009); Kober et al (2010); Pastor et al (2011); Peters and Schaal (2008a); Peters et al (2010a)

methods optimize locally around existing policies π_i by computing policy changes $\delta\pi_i$ that will increase the expected return and results in iterative updates in the form

$$\pi_{i+1} = \pi_i + \delta\pi_i.$$

The computation of the policy update is the key step here and a variety of updates have been proposed ranging from pairwise comparisons (Strens and Moore, 2001; Ng et al, 2004a) over gradient estimation using finite policy differences (Geng et al, 2006; Mitsunaga et al, 2005; Sato et al, 2002; Tedrake et al, 2005) and heuristic parallel search methods (such as genetic algorithms, see (Goldberg, 1989)) to approaches coming from optimal control such as differential dynamic programming (DDP) (Atkeson, 1998) and multiple shooting approaches (Betts, 2001) as well as core reinforcement learning methods.

In recent years, general reinforcement learning has yielded three kinds of policy search approaches that have translated particularly well into the domain of robotics: (i) policy gradients approaches based on likelihood-ratio estimation (Sutton et al, 2000), (ii) policy updates inspired by expectation maximization (Toussaint et al, 2010), and (iii) the path integral methods (Kappen, 2005). Likelihood-ratio policy gradient methods rely on perturbing the motor command instead of comparing in policy space. Initial approaches such as REINFORCE (Williams, 1992) have been rather slow but recent natural policy gradient approaches (Peters and Schaal, 2008c,b) have allowed faster convergence which may be useful for robotics. When the reward is treated as an improper probability distribution (Dayan and Hinton, 1997), safe and fast methods can be derived that are inspired by expectation-maximization. Some of these approaches have proven successful in robotics, e.g., reward-weighted regression (Peters and Schaal, 2008a), PoWER (Kober and Peters, 2009), MCEM (Vlassis et al, 2009) and cost-regularized kernel regression (Kober et al, 2010). The path integral methods have yielded policy updates vaguely related to PoWER such as PI² (Theodorou et al, 2010) which are very promising for robotics. An overview of publications using policy search methods is presented in Table 18.2.

18.4 Tractability through Representation

Much of the success of reinforcement learning methods has been due to the smart use of approximate representations. In a domain that is so inherently beyond the reach of complete tabular representation, the need of such approximations is particularly pronounced. The different ways of making reinforcement learning methods tractable in robotics are tightly coupled to the underlying framework. Policy search methods require a choice of policy representation that limits the number of representable policies to enhance learning speed, see Section 18.4.3. A value function-based approach requires an accurate, robust but general function approximator that can capture the value function sufficiently precisely, see Section 18.4.2. Reducing the dimensionality of states or actions by smart state-action discretization is a representational simplification that may enhance both policy search and value

function-based methods, see 18.4.1. An overview of publications using representations to render the learning problem tractable is presented in Table 18.3.

18.4.1 Smart State-Action Discretization

Having lower-dimensional states or actions eases most reinforcement learning problems significantly, particularly in the context of robotics. Here, we give a quick overview of different attempts to achieve this goal with smart discretization.

Hand Crafted. A variety of authors have manually tuned discretizations so that basic tasks can be learned on real robots. For low-dimensional tasks, such as balancing a ball on a beam (Benbrahim et al, 1992), we can generate discretizations straightforwardly while much more human experience is needed for more complex tasks. Such tasks range from basic navigation with noisy sensors (Willgoss and Iqbal, 1999) over one degree of freedom ball-in-a-cup (Nemec et al, 2010), two degree of freedom crawling motions (Tokic et al, 2009), and learning object affordances (Paletta et al, 2007) up to gait patterns for four legged walking (Kimura et al, 2001).

Learned from Data. Instead of specifying the discretizations by hand, they can also be learned from data. For example, a rule based reinforcement learning approach automatically segmented the state space to learn a cooperative task with mobile robots (Yasuda and Ohkura, 2008). In the related field of computer vision, Piater et al (2010) propose an approach that adaptively and incrementally discretizes a perceptual space into discrete states.

Meta-Actions. Automatic construction of meta actions has fascinated reinforcement learning researchers and there are various examples in the literature. For example, in (Asada et al, 1996), the state and action sets are constructed in a way that repeated action primitives leads to a change in state to overcome problems associated with the discretization. Q -learning and dynamic programming based approaches have been compared in a pick-n-place task (Kalmár et al, 1998) using modules. A task of transporting a ball with a dog robot (Soni and Singh, 2006) can be learned with semi-automatically discovered options. Using only the sub-goals of primitive motions, a pouring task can be learned by a humanoid robot (Nemec et al, 2009). Various other examples range from foraging (Mataric, 1997) and cooperative tasks (Mataric, 1994) with multiple robots, to grasping with restricted search spaces (Platt et al, 2006) navigation of a mobile robot (Dorigo and Colombetti, 1993). These approaches belong to hierarchical reinforcement learning approaches discussed in more detail in Chapter 9.

Relational Representation. In a relational representation the states, actions, and transitions are not represented individually but entities of the same, predefined type are grouped and their relations are considered. This representation has been employed to learn to navigate buildings with a real robot in a supervised setting (Cocora et al, 2006) and to manipulate articulated objects in simulation (Katz et al, 2008).

Table 18.3 This table illustrates different methods of making robot reinforcement learning tractable by employing a suitable representation

SMART STATE-ACTION DISCRETIZATION

Approach	Employed by...
<i>Hand crafted</i>	Benbrahim et al (1992); Kimura et al (2001); Nemeć et al (2010); Paletta et al (2007); Tokic et al (2009); Willgoss and Iqbal (1999)
<i>Learned</i>	Piater et al (2010); Yasuda and Ohkura (2008)
<i>Meta-actions</i>	Asada et al (1996); Dorigo and Colombetti (1993); Kalmár et al (1998); Mataric (1994, 1997); Platt et al (2006); Soni and Singh (2006); Nemeć et al (2009)
<i>Relational Representation</i>	Cocora et al (2006); Katz et al (2008)

FUNCTION APPROXIMATION

Approach	Employed by...
<i>Local Models</i>	Bentivegna (2004); Schaal (1997); Smart and Kaelbling (1998)
<i>Neural Networks</i>	Benbrahim and Franklin (1997); Duan et al (2008); Gaskett et al (2000); Hafner and Riedmiller (2003); Riedmiller et al (2009); Thrun (1995)
<i>GPR</i>	Gräve et al (2010); Kroemer et al (2009, 2010); Lizotte et al (2007); Rottmann et al (2007)
<i>Neighbors</i>	Hester et al (2010); Mahadevan and Connell (1992); Touzet (1997)

PRE-STRUCTURED POLICIES

Approach	Employed by...
<i>Motor Primitives</i>	Kohl and Stone (2004); Kober and Peters (2009); Peters and Schaal (2008c,b); Theodorou et al (2010)
<i>Neural Networks</i>	Endo et al (2008); Geng et al (2006); Gullapalli et al (1994); Hailu and Sommer (1998)
<i>Via Points</i>	Miyamoto et al (1996)
<i>Linear Models</i>	Tamei and Shibata (2009)
<i>GMM & LLM</i>	Deisenroth and Rasmussen (2010); Guenter et al (2007); Peters and Schaal (2008a)
<i>Controller</i>	Kolter and Ng (2009); Tedrake (2004); Tedrake et al (2005); Vlassis et al (2009)
<i>Non-parametric</i>	Kober et al (2010); Mitsunaga et al (2005); Peters et al (2010a)

18.4.2 Function Approximation

Function approximation has always been the key component that allowed value function methods to scale into interesting domains. In robot reinforcement learning, the following function approximation schemes have been popular and successful.

Neural networks. Neural networks as function approximators for continuous states and actions have been used by various groups, e.g., multi-layer perceptrons were used to learn a wandering behavior and visual servoing (Gaskell et al, 2000); fuzzy neural networks (Duan et al, 2008) as well as explanation-based neural networks (Thrun, 1995) have allowed learning basic navigation while CMAC neural networks have been used for biped locomotion (Benbrahim and Franklin, 1997). A particularly impressive application has been the success of Brainstormers RoboCup soccer team that used multi-layer perceptrons for winning the world cup several times in different leagues (Hafner and Riedmiller, 2003; Riedmiller et al, 2009).

Generalize to Neighboring Cells. As neural networks are globally affected from local errors, much work has focused on simply generalizing from neighboring cells. One of the earliest papers in robot reinforcement learning (Mahadevan and Connell, 1992) introduced this idea by statistical clustering states to speed up a box pushing task with a mobile robot, see Figure 18.1a. This approach was also used for a navigation and obstacle avoidance task with a mobile robot (Touzet, 1997). Similarly, decision trees have been used to generalize states and actions to unseen ones, e.g., to learn a penalty kick on a humanoid robot (Hester et al, 2010).

Local Models. Locally weighted regression is known to be a particularly efficient function approximator. Using it for value function approximation has allowed learning a navigation task with obstacle avoidance (Smart and Kaelbling, 1998), a pole balancing task Schaal (1997) as well as an air hockey task (Bentivegna, 2004).

Gaussian Process Regression. Using GPs as function approximator for the value function has allowed learning of hovering with an autonomous blimp (Rottmann et al, 2007), see Figure 18.1c. Similarly, another paper shows that grasping can be learned using Gaussian Process Regression (Gräve et al, 2010). Grasping locations can be learned by focusing on rewards, modeled by GPR, by trying candidates with predicted high rewards (Kroemer et al, 2009). High reward uncertainty allows intelligent exploration in reward-based grasping (Kroemer et al, 2010). Gait of robot dogs can be optimized by first learning the expected return function with a Gaussian process regression and subsequently searching for the optimal solutions (Lizotte et al, 2007).

18.4.3 Pre-structured Policies

To make the policy search approach tractable, the policy needs to be represented with an appropriate function approximation.

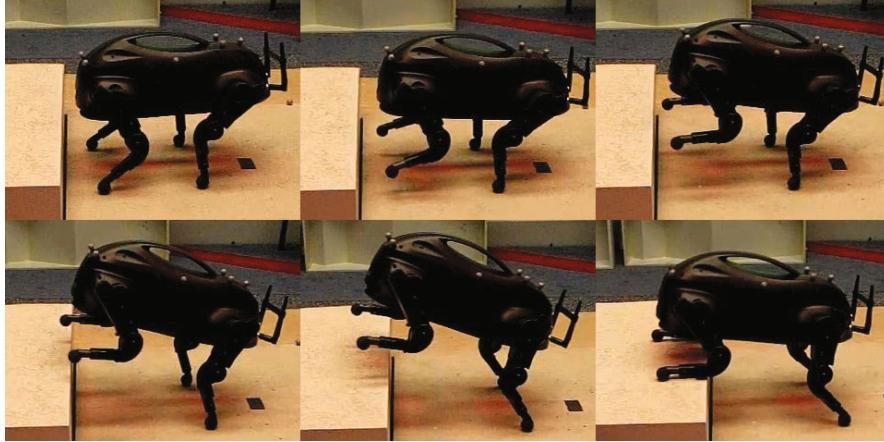


Fig. 18.3 Boston Dynamics LittleDog jumping (Kolter and Ng, 2009) (Picture reprinted with permission of Zico Kolter)

Motor Primitives. Motor primitives are a biologically-inspired concept and represent basic movements. For discrete movements the dynamical system motor primitives (Ijspeert et al, 2003; Schaal et al, 2007) representation has been employed to learn a T-ball batting task (Peters and Schaal, 2008c,b), the underactuated swing-up and ball-in-a-cup (Kober and Peters, 2009), flipping a light switch (Buchli et al, 2011), as well as playing pool and manipulating a box (Pastor et al, 2011). For rhythmic behaviors half-elliptical locuses have been used as a representation of the gait pattern of a robot dog (Kohl and Stone, 2004).

Neural Networks. Instead of analytically describing rhythmic movements, neural networks can be used as oscillators to learn gaits of a two legged robot (Geng et al, 2006; Endo et al, 2008). Also a peg-in-hole (see Figure 18.1b) and a ball-balancing task (Gullapalli et al, 1994) as well as a navigation task (Hailu and Sommer, 1998) have been learned with neural networks as policy function approximators.

Via Points. Optimizing the position and timing of via-points Miyamoto et al (1996) learned a kendama task.

Linear Models. Tamei and Shibata (2009) used reinforcement learning to adjust a model that maps from EMG signals to forces that in turn is used in a cooperative holding task.

Gaussian Mixture Models and Locally Linear Models. One of the most general function approximators is based on radial basis functions, also called Gaussian kernels. However, specifying the centers and widths of these is challenging. These locations and variances can also be estimated from data prior to the reinforcement learning process which has been used to generalize a reaching movement (Guenter et al, 2007) and to learn the cart-pole task (Deisenroth and Rasmussen, 2010).

Operational space control was learned by Peters and Schaal (2008a) using locally linear models.

Controller. Here, parameters of a local linear controller are learned. Applications include learning to walk in 20 minutes with a biped robot (Tedrake, 2004; Tedrake et al, 2005), to drive a radio-controlled (RC) car as well as a jumping behavior for a robot dog jump (Kolter and Ng, 2009), as illustrated in Figure 18.3, and to balance a two wheeled robot (Vlassis et al, 2009).

Non-parametric. Also in this context non-parametric representations can be used. The weights of different robot human interaction possibilities (Mitsunaga et al, 2005), the weights of different striking movements in a table tennis task (Peters et al, 2010a), and the parameters of meta-actions for dart and table tennis tasks (Kober et al, 2010) can be optimized.

18.5 Tractability through Prior Knowledge

Prior knowledge can significantly help to guide the learning process. Prior knowledge can be included in the form of initial policies, initial models, or a predefined structure of the task. These approaches significantly reduce the search space and, thus, speed up the learning process. Providing a goal achieving initial policy allows a reinforcement learning method to quickly explore promising regions in the value functions or in policy space, see Section 18.5.1. Pre-structuring the task to break a complicated task down into several more tractable ones can be very successful, see Section 18.5.2. An overview of publications using prior knowledge to render the learning problem tractable is presented in Table 18.4.

18.5.1 Prior Knowledge through Demonstrations

Animals and humans frequently learn using a combination of imitation and trial and error. For example, when learning to play tennis, an instructor usually shows the student how to do a proper swing, e.g., a forehand or backhand. The student will subsequently imitate this behavior but still needs hours of practicing to successfully return balls to the opponent's court. Input from a teacher is not limited to initial instruction. The instructor can give additional demonstrations in a later learning stage (Latzeke et al, 2007), these can also be used as differential feedback (Argall et al, 2008). Global exploration is not necessary as the student can improve by locally optimizing his striking movements previously obtained by imitation. A similar approach can speed up robot reinforcement learning based on human demonstrations or initial hand coded policies. For a recent survey on imitation learning for robotics see (Argall et al, 2009).

Demonstrations by a Teacher. Demonstrations can be obtained by remote controlling the robot, which was used to initialize a Q -table for a navigation task (Conn and Peters II, 2007). If the robot is back-drivable, kinesthetic teach-in (i.e., by

Table 18.4 This table illustrates different methods of making robot reinforcement learning tractable by incorporating prior knowledge

DEMONSTRATION	
Approach	Employed by...
<i>Teacher</i>	Bitzer et al (2010); Conn and Peters II (2007); Gräve et al (2010); Kober et al (2008); Kober and Peters (2009); Latzke et al (2007); Peters and Schaal (2008c,b)
<i>Policy</i>	Birdwell and Livingston (2007); Erden and Leblebicioaglu (2008); Martínez-Marín and Duckett (2005); Smart and Kaelbling (1998); Tedrake (2004); Tedrake et al (2005); Wang et al (2006)
TASK STRUCTURE	
Approach	Employed by...
<i>Hierarchical</i>	Donnart and Meyer (1996); Kirchner (1997); Morimoto and Doya (2001)
<i>Progressive Tasks</i>	Asada et al (1996)
DIRECTED EXPLORATION	
	Employed by...
	Huang and Weng (2002); Kroemer et al (2010); Pendrith (1999)

taking it by the hand and moving it) can be employed. This method has resulted in applications including T-ball batting (Peters and Schaal, 2008c,b), reaching tasks (Guenter et al, 2007; Bitzer et al, 2010), ball-in-a-cup (Kober and Peters, 2009), flipping a light switch (Buchli et al, 2011), as well as playing pool and manipulating a box (Pastor et al, 2011). Motion-capture setups can be used alternatively, but the demonstrations are often not as informative due to the correspondence problem. Demonstrations obtained by motion capture have been used to learn ball-in-a-cup (Kober et al, 2008) and grasping (Gräve et al, 2010).

Hand Coded Policy. A pre-programmed policy can provide demonstrations instead of a human teacher. A vision-based mobile robot docking task can be learned faster with such a basic behavior than using Q -learning alone as demonstrated in (Martínez-Marín and Duckett, 2005). As an alternative corrective actions when the robot deviates significantly from the desired behavior can be employed as prior knowledge. This approach has been applied to adapt walking patterns of a robot dog to new surfaces (Birdwell and Livingston, 2007) by Q -learning. Having hand-coded stable initial gaits can significantly help as shown on six-legged robot gait (Erden and Leblebicioaglu, 2008) as well as on a biped (Tedrake, 2004; Tedrake et al, 2005).

18.5.2 *Prior Knowledge through Task Structuring*

Often a task can be decomposed hierarchically into basic components (see Chapter 9) or in a sequence of increasingly difficult tasks. In both cases the complexity of the learning task is significantly reduced.

Hierarchical Reinforcement Learning. Easier tasks can be used as building blocks for a more complex behavior. For example, hierarchical Q -learning has been used to learn different behavioral levels for a six legged robot: moving single legs, locally moving the complete body, and globally moving the robot towards a goal (Kirchner, 1997). A stand-up behavior considered as a hierarchical reinforcement learning task has been learned using Q -learning in the upper-level and TD-learning in the lower level (Morimoto and Doya, 2001). Navigation in a maze can be learned using an actor-critic architecture by tuning the influence of different control modules and learning these modules (Donnart and Meyer, 1996).

Progressive Tasks. Often complicated tasks are easier to learn if simpler tasks can already be performed. A sequence of increasingly difficult missions has been employed to learn a goal shooting task in (Asada et al, 1996) using Q -learning.

18.5.3 *Directing Exploration with Prior Knowledge*

A mobile robot learns to direct attention (Huang and Weng, 2002) by employing a modified Q -learning approach using novelty. Using ‘corrected truncated returns’ and taking into account the estimator variance, a six legged robot employed with stepping reflexes can learn to walk (Pendrith, 1999). Using upper confidence bounds to direct exploration grasping can be learned efficiently (Kroemer et al, 2010). Offline search can be used to enhance Q -learning during a grasping task (Wang et al, 2006).

18.6 Tractability through Simulation

Using a simulation instead of the real physical robot has major advantages such as safety and speed. A simulation can be used to eliminate obviously bad behaviors and often runs much faster than real time. Simulations are without doubt a helpful testbed for debugging algorithms. A popular approach is to combine simulations and real evaluations by only testing promising policies on the real system and using it to collect new data to refine the simulation (Section 18.6.2). Unfortunately, directly transferring policies learned in simulation to a real system can be challenging (Section 18.6.3). An overview of publications using simulations to render the learning problem tractable is presented in Table 18.5.

Table 18.5 This table illustrates different methods of making robot reinforcement learning tractable using simulations

SIMULATIONS	
Approach	Employed by...
<i>Mental Rehearsal</i>	Abbeel et al (2006); Atkeson and Schaal (1997); Atkeson et al (1997); Atkeson (1998); Bagnell and Schneider (2001); Bakker et al (2006); Coates et al (2009); Deisenroth and Rasmussen (2010); Ko et al (2007); Kolter et al (2008); Michels et al (2005); Nemeć et al (2010); Ng et al (2004b,a); Schaal and Atkeson (1994); Uchibe et al (1998)
<i>Direct Policy Transfer</i>	Bakker et al (2003); Duan et al (2007); Fagg et al (1998); Ilg et al (1999); Oßwald et al (2010); Svinin et al (2001); Youssef (2005)



Fig. 18.4 Autonomous inverted helicopter flight (Ng et al, 2004b)(Picture reprint with permission of Andrew Ng)

18.6.1 Role of Models

Model-free algorithms try to directly learn the value function or the policy. Model-based approaches jointly learn a model of the system and the value function or the policy. Model-based methods can make the learning process a lot more sample efficient. However, depending on the type of model these may require a lot of memory. Model-based approaches rely on an approach that finds good policies in the model. These methods encounter the risk of exploiting model inaccuracies to decrease the cost. If the learning methods require predicting the future or using derivatives, the inaccuracies may accumulate quickly, and, thus, significantly amplify noise and errors. These effects lead to value functions or policies that work well in the model but poorly on the real system. This issue is highly related to the transfer problem discussed in Section 18.2.4. A solution is to overestimate the noise, to introduce a controlled amount of inconsistency (Atkeson, 1998), or to use a crude model to find a policy that compensates the derivative of the behavior in the model and on the real system (Abbeel et al, 2006). See Chapter 4 for a more detailed discussion.

In Section 18.2.4, we discussed that policies learned in simulation often cannot be transferred to the real system. However, simulations are still a very useful tool.

Most simulations run significantly faster than real time and many problems associated with expensive samples (Section 18.2.2) can be avoided. For these reasons simulations are usually used to debug, test and optimize algorithms. Learning in simulation often can be made significantly easier than on real robots. The noise can be controlled and all variables can be accessed. If the approach does not work in simulation it is often unlikely that it works on the real system. Many papers also use simulations to benchmark approaches as repeating the experiment frequently to observe the average behavior and to compare many algorithms is often not feasible on the real system.

18.6.2 Mental Rehearsal

The idea of combining learning in simulation and in the real environment has been popularized by the Dyna-architecture (Sutton, 1990) in reinforcement learning. Due to the obvious advantages in the robotics domain, it has been proposed in this context as well. Experience collected in the real world can be used to learn a forward model (rAström and Wittenmark, 1989) from data. Such a forward model allows training in a simulated environment and the resulting policy is subsequently transferred to the real environment. This approach can also be iterated and may significantly reduce the needed interactions with the real world. However, often the learning process can exploit the model errors which may lead to biased solutions and slow convergence.

Such mental rehearsal has found a lot of applications in robot reinforcement learning. Parallel learning in simulation and directed exploration allows Q -learning to learn a navigation task from scratch in 20 minutes (Bakker et al, 2006). Two robots taking turns in learning a simplified soccer task were also able to profit from mental rehearsal (Uchibe et al, 1998). Atkeson et al (1997) learned a billiard and a devil sticking task employing forward models. Nemec et al (2010) used a value function learned in simulation to initialize the real robot learning.

To reduce the simulation bias resulting from model errors, Ng et al (2004b,a) suggested re-using the series of random numbers in learned simulators for robotics and called this approach PEGASUS. Note, that this approach is well-known in the simulation community (Glynn, 1987) with fixed models. The resulting approach found various applications in the learning artistic maneuvers for autonomous helicopters (Bagnell and Schneider, 2001; Ng et al, 2004b,a), as illustrated in Figure 18.4, it has been used to learn control parameters for a RC car (Michels et al, 2005) and an autonomous blimp (Ko et al, 2007). Alternative means to use crude models by grounding policies in real world data have been suggested in (Abbeel et al, 2006) and were employed to learn steering a RC car. Instead of sampling from a forward model-based simulator, such learned models can also be directly used for computing optimal control policies. This has resulted in a variety of robot reinforcement learning applications ranging from pendulum swing-up tasks learned with DDP (Atkeson and Schaal, 1997; Atkeson, 1998), devil-sticking (a form of gyroscopic

juggling) obtained with local LQR solutions (Schaal and Atkeson, 1994), trajectory following with space-indexed controllers trained with DDP for an autonomous RC car (Kolter et al, 2008), the cart-pole task (Deisenroth and Rasmussen, 2010), to aerobatic helicopter flight trained with DDP (Coates et al, 2009). Solving an LQR problem with multiple probabilistic models and combining the resulting closed-loop control with open-loop control has resulted in autonomous sideways sliding into a parking spot (Kolter et al, 2010). A promising new related approach are LQR-trees (Tedrake et al, 2010).

18.6.3 Direct Transfer from Simulated to Real Robots

Only few papers claim that a policy learned in simulation can directly be transferred to a real robot while maintaining its high level of performance. The few examples include maze navigation tasks (Bakker et al, 2003; Oßwald et al, 2010; Youssef, 2005) and obstacle avoidance (Fagg et al, 1998) for a mobile robot. Similar transfer was achieved in very basic robot soccer (Duan et al, 2007) and multi-legged robot locomotion (Ilg et al, 1999; Svinin et al, 2001).

18.7 A Case Study: Ball-in-a-Cup

Up to this point in this book chapter, we have reviewed a large variety of problems and associated possible solutions of robot reinforcement learning. In this section, we will take an orthogonal approach and discuss one task in detail that we have previously studied. This task is called ball-in-a-cup and due to its complexity, it can serve as an example to highlight some of the various discussed challenges and methods. In Section 18.7.1, we describe the experimental setting with a focus on the task and the reward. Section 18.7.2 discusses a type of pre-structured policies that has been particularly useful in robotics. Inclusion of prior knowledge is presented in Section 18.7.3. We explain the advantages of the employed policy search algorithm in Section 18.7.4. We will discuss the use of simulations in this task in Section 18.7.5. Finally, we explore an alternative reinforcement learning approach in Section 18.7.6.

18.7.1 Experimental Setting: Task and Reward

The children’s motor game ball-in-a-cup, also known as balero and bilboquet, is challenging even for most adults. The toy consists of a small cup held in one hand (or, in our case, is attached to the end-effector of the robot) and a small ball is hanging on a string attached to the cup’s bottom (for our toy, the string is 40cm long). Initially, the ball is at rest, hanging down vertically. The player needs to move fast to induce motion in the ball through the string, toss it up and catch it with the

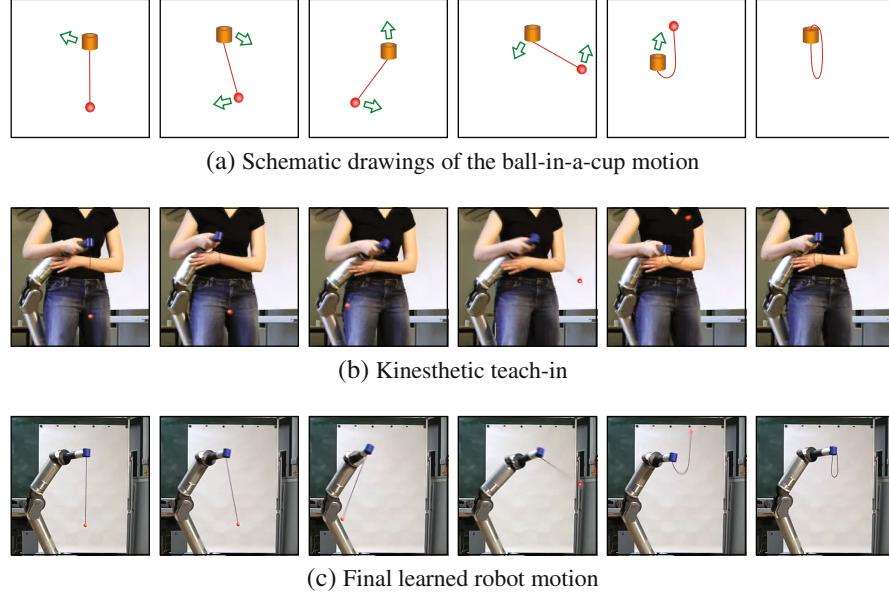


Fig. 18.5 This figure shows schematic drawings of the ball-in-a-cup motion (a), the final learned robot motion (c), as well as a kinesthetic teach-in (b). The green arrows show the directions of the current movements in that frame. The human cup motion was taught to the robot by imitation learning with 31 parameters per joint for an approximately 3 seconds long movement. The robot manages to reproduce the imitated motion quite accurately, but the ball misses the cup by several centimeters. After approximately 75 iterations of our Policy learning by Weighting Exploration with the Returns (PoWER) algorithm the robot has improved its motion so that the ball regularly goes into the cup. See Figure 18.6 for the performance increase.

cup. A possible movement is illustrated in Figure 18.5a. As the string frequently entangles in failures and the robot cannot unravel it, human intervention is required and an automatic reset is often not possible.

The state of the system can be described by joint angles and joint velocities of the robot as well as the the Cartesian coordinates and velocities of the ball. The actions are the joint space accelerations (which are translated into torques by an inverse dynamics controller). Thus, we have to deal with twenty state and seven action dimensions. Discretizing the state-action space for a value function based approach is not likely to be feasible.

At the time t_c where the ball passes the rim of the cup with a downward direction, we compute the reward as $r(t_c) = \exp(-\alpha(x_c - x_b)^2 - \alpha(y_c - y_b)^2)$ while we have $r(t) = 0$ for all $t \neq t_c$. Here, the cup position is denoted by $[x_c, y_c, z_c] \in \mathbb{R}^3$, the ball position $[x_b, y_b, z_b] \in \mathbb{R}^3$ and we have a scaling parameter $\alpha = 100$. Initially, we used a reward function based solely on the minimal distance between the ball and the cup. However, the algorithm has exploited rewards resulting from hitting the cup

with the ball from below or from the side as such behaviors are easier to achieve and yield comparatively high rewards. To avoid such local optima, it was essential to find a good reward function such as the initially described one.

The task is quite complex as the reward is not only affected by the cup's movements but foremost by the ball's movements. As the ball's movements are very sensitive to small perturbations, the initial conditions or small arm movement changes, will drastically affect the outcome. Creating an accurate simulation is hard due to the nonlinear, unobservable dynamics of the string and its non-negligible weight.

18.7.2 Appropriate Policy Representation

The policy is represented by dynamical system motor primitives (Ijspeert et al, 2003; Schaal et al, 2007). The global movement is encoded as a point attractor linear dynamical system. The details of the movement are generated by a transformation function that allows learning complex behaviors. This transformation function is modeled using locally linear function approximation. This combination of the global attractor behavior and local transformation allows a very parsimonious representation of the policy. This policy is linear in parameters $a = \theta\mu(s)$ and, thus, it is straightforward to include prior knowledge from a demonstration using supervised learning by locally weighted regression.

18.7.3 Generating a Teacher's Demonstration

Due to the complexity of the task, ball-in-a-cup is a hard motor task even for children who usually only succeed after observing another person presenting a demonstration, and require additional trial-and-error-based learning. Mimicking how children learn ball-in-a-cup, we first initialize the motor primitives by imitation and, subsequently, improve them by reinforcement learning.

We obtained a demonstration for imitation by recording the motions of a human player performing kinesthetic teach-in as shown in Figure 18.5b. Kinesthetic teach-in means ‘taking the robot by the hand’, performing the task by moving the robot while it is in gravity-compensation mode and recording the joint angles, velocities and accelerations. It requires a back-drivable robot system similar to a human arm. From the imitation, the number of needed policy parameters can be determined by cross-validation. As the robot fails to catch the ball with the cup, additional reinforcement learning is needed for self-improvement.

18.7.4 Reinforcement Learning by Policy Search

When learning motor primitives, we learn a deterministic mean policy $\bar{a} = \theta\mu(s) = f(z)$ which is linear in parameters θ and augmented by additive exploration $\varepsilon(s,t)$ to make model-free reinforcement learning possible. As a result, the explorative policy

can be given in the form $a = \theta\mu(s,t) + \varepsilon(\mu(s,t))$. Policy search approaches often focus on state-independent, white Gaussian exploration, i.e., $\varepsilon(\mu(s,t)) \sim \mathcal{N}(0, \Sigma)$ which has resulted into applications such as T-Ball batting (Peters and Schaal, 2008c) and constrained movement (Guenter et al, 2007). However, from our experience, such unstructured exploration at every step has several disadvantages, i.e., (i) it causes a large variance which grows with the number of time-steps, (ii) it perturbs actions too frequently, thus, ‘washing’ out their effects and (iii) can damage the system executing the trajectory.

Alternatively, one could generate a form of structured, state-dependent exploration (Rückstieß et al, 2008) $\varepsilon(\mu(s,t)) = \varepsilon_t\mu(s,t)$ with $[\varepsilon_t]_{ij} \sim \mathcal{N}(0, \sigma_{ij}^2)$, where σ_{ij}^2 are meta-parameters of the exploration that can also be optimized. This argument results into the policy $a \sim \pi(a_t|s_t, t) = \mathcal{N}(a|\mu(s,t), \hat{\Sigma}(s,t))$.

In (Kober and Peters, 2009), we have derived a framework of reward weighted imitation. Based on (Dayan and Hinton, 1997) we consider the return of an episode as an improper probability distribution. We maximize a lower bound of the logarithm of the expected return. Depending on the strategy of optimizing this lower bound and the exploration strategy, the framework yields several well known policy search algorithms: episodic REINFORCE (Williams, 1992), the policy gradient theorem (Sutton et al, 2000), episodic natural actor critic (Peters and Schaal, 2008b), a generalization of the reward-weighted regression (Peters and Schaal, 2008a) as well as our novel Policy learning by Weighting Exploration with the Returns (PoWER) algorithm. PoWER is an expectation-maximization inspired algorithm that employs state-dependent exploration. The update rule is given by

$$\theta' = \theta + \frac{E_\tau \left\{ \sum_{t=1}^T \varepsilon_t Q^\pi(s_t, a_t, t) \right\}}{E_\tau \left\{ \sum_{t=1}^T Q^\pi(s_t, a_t, t) \right\}}.$$

To reduce the number of trials in this on-policy scenario, we reuse the trials through importance sampling (Sutton and Barto, 1998). To avoid the fragility sometimes resulting from importance sampling in reinforcement learning, samples with very small importance weights are discarded. This algorithm performs basically a local search around the policy learned from prior knowledge.

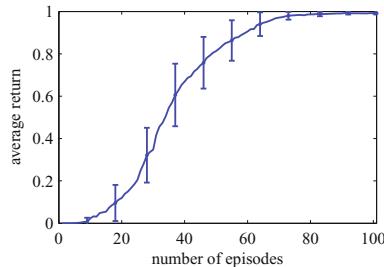


Fig. 18.6 This figure shows the expected return of the learned policy in the ball-in-a-cup evaluation averaged over 20 runs

Figure 18.6 shows the expected return over the number of episodes where convergence to a maximum is clearly recognizable. The robot regularly succeeds at bringing the ball into the cup after approximately 75 episodes.

Using a value function based approach would require an unrealistic amount of samples to get a good estimate of the value function. Greedily searching for an optimal motor command in such a high-dimensional action space is probably as hard as finding a locally optimal policy.

18.7.5 Use of Simulations in Robot Reinforcement Learning

We created a simulation of the robot using rigid body dynamics with parameters estimated from data. The toy is simulated as a pendulum with an elastic string that switches to a ballistic point mass when the ball is closer to the cup than the string is long. The spring, damper and restitution constants were tuned to match recorded data. Even though this simulation matches recorded data very well, policies that get the ball in the cup in simulation usually miss the cup by several centimeters on the real system and vice-versa. However, this simulation was very helpful to develop and tune the algorithm as it runs faster in simulation than real-time and does not require human supervision or intervention.

18.7.6 Alternative Approach with Value Function Methods

Nemec et al (2010) used a different reinforcement learning approach to achieve the ball-in-a-cup task with a Mitsubishi PA10 robot. They decomposed the task in two sub-tasks, the swing-up phase and the catching phase. In the swing-up phase the ball is moved above the cup. In the catching phase the ball is caught with the cup using an analytic prediction of the ball trajectory based on the movement of a flying point mass. The catching behavior is fixed, only the swing-up behavior is learned. The paper proposes to use SARSA to learn the swing-up movement. The states consist of the cup positions and velocities as well as the angular positions and velocities of the ball. The actions are the accelerations of the cup in a single Cartesian direction. Tractability is achieved by discretizing both the states (324 values) and the actions (5 values) and initialization by simulation. The behavior was first learned in simulation requiring 220 to 300 episodes. The state-action value function learned in simulation was used to initialize the learning on the real robot. The robot required an additional 40 to 90 episodes to adapt the behavior learned in simulation to the real environment.

18.8 Conclusion

In this chapter, we have surveyed robot reinforcement learning in order to introduce general reinforcement learning audiences to the state of the art in this domain.

We have pointed out the inherent challenges such as the high-dimensional continuous state and action space, the high cost associated with trials, the problems associated with transferring policies learned in simulation to real robots as well as the need for appropriate reward functions. A discussion of how different robot reinforcement learning approaches are affected by the domain has been given. We have surveyed different authors' approaches to render robot reinforcement learning tractable through improved representation, inclusion of prior knowledge and usage of simulation. To highlight aspects that we found particularly important, we give a case study on how a robot can learn a complex task such as ball-in-a-cup.

References

- Abbeel, P., Quigley, M., Ng, A.Y.: Using inaccurate models in reinforcement learning. In: International Conference on Machine Learning, ICML (2006)
- Abbeel, P., Coates, A., Quigley, M., Ng, A.Y.: An application of reinforcement learning to aerobatic helicopter flight. In: Advances in Neural Information Processing Systems, NIPS (2007)
- Abbeel, P., Dolgov, D., Ng, A.Y., Thrun, S.: Apprenticeship learning for motion planning with application to parking lot navigation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2008)
- Argall, B.D., Browning, B., Veloso, M.: Learning robot motion control with demonstration and advice-operators. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2008)
- Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57, 469–483 (2009)
- Asada, M., Noda, S., Tawaratsumida, S., Hosoda, K.: Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning* 23(2-3), 279–303 (1996)
- Atkeson, C., Moore, A., Stefan, S.: Locally weighted learning for control. *AI Review* 11, 75–113 (1997)
- Atkeson, C.G.: Using local trajectory optimizers to speed up global optimization in dynamic programming. In: Advances in Neural Information Processing Systems, NIPS (1994)
- Atkeson, C.G.: Nonparametric model-based reinforcement learning. In: Advances in Neural Information Processing Systems, NIPS (1998)
- Atkeson, C.G., Schaal, S.: Robot learning from demonstration. In: International Conference on Machine Learning, ICML (1997)
- Bagnell, J.A., Schneider, J.C.: Autonomous helicopter control using reinforcement learning policy search methods. In: IEEE International Conference on Robotics and Automation, ICRA (2001)
- Bakker, B., Zhumatiy, V., Gruener, G., Schmidhuber, J.: A robot that reinforcement-learns to identify and memorize important previous observations. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2003)
- Bakker, B., Zhumatiy, V., Gruener, G., Schmidhuber, J.: Quasi-online reinforcement learning for robots. In: IEEE International Conference on Robotics and Automation, ICRA (2006)
- Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13(4), 341–379 (2003)

- Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton (1957)
- Bellman, R.E.: Introduction to the Mathematical Theory of Control Processes, vol. 40-I. Academic Press, New York (1967)
- Bellman, R.E.: Introduction to the Mathematical Theory of Control Processes, vol. 40-II. Academic Press, New York (1971)
- Benbrahim, H., Franklin, J.A.: Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems* 22(3-4), 283–302 (1997)
- Benbrahim, H., Doleac, J., Franklin, J., Selfridge, O.: Real-time learning: a ball on a beam. In: International Joint Conference on Neural Networks, IJCNN (1992)
- Bentivegna, D.C.: Learning from observation using primitives. PhD thesis, Georgia Institute of Technology (2004)
- Betts, J.T.: Practical methods for optimal control using nonlinear programming. In: Advances in Design and Control, vol. 3. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (2001)
- Birdwell, N., Livingston, S.: Reinforcement learning in sensor-guided aibo robots. Tech. rep., University of Tennessee, Knoxville, advised by Dr. Itamar Elhanany (2007)
- Bitzer, S., Howard, M., Vijayakumar, S.: Using dimensionality reduction to exploit constraints in reinforcement learning. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2010)
- Buchli, J., Stulp, F., Theodorou, E., Schaal, S.: Learning variable impedance control. *International Journal of Robotics Research Online First* (2011)
- Bušoniu, L., Babuška, R., De Schutter, B., Ernst, D.: Reinforcement Learning and Dynamic Programming Using Function Approximators. CRC Press, Boca Raton (2010)
- Coates, A., Abbeel, P., Ng, A.Y.: Apprenticeship learning for helicopter control. *Commun. ACM* 52(7), 97–105 (2009)
- Cocora, A., Kersting, K., Plagemann, C., Burgard, W., Raedt, L.D.: Learning relational navigation policies. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2006)
- Conn, K., Peters II, R.A.: Reinforcement learning with a supervisor for a mobile robot in a real-world environment. In: IEEE International Symposium on Computational Intelligence in Robotics and Automation, CIRA (2007)
- Dayan, P., Hinton, G.E.: Using expectation-maximization for reinforcement learning. *Neural Computation* 9(2), 271–278 (1997)
- Deisenroth, M.P., Rasmussen, C.E.: A practical and conceptual framework for learning in control. Tech. Rep. UW-CSE-10-06-01, Department of Computer Science & Engineering, University of Washington, USA (2010)
- Donnart, J.Y., Meyer, J.A.: Learning reactive and planning rules in a motivationally autonomous animat. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 26(3), 381–395 (1996)
- Dorigo, M., Colombetti, M.: Robot shaping: Developing situated agents through learning. Tech. rep., International Computer Science Institute, Berkeley, CA (1993)
- Duan, Y., Liu, Q., Xu, X.: Application of reinforcement learning in robot soccer. *Engineering Applications of Artificial Intelligence* 20(7), 936–950 (2007)
- Duan, Y., Cui, B., Yang, H.: Robot Navigation Based on Fuzzy RL Algorithm. In: Sun, F., Zhang, J., Tan, Y., Cao, J., Yu, W. (eds.) ISNN 2008, Part I. LNCS, vol. 5263, pp. 391–399. Springer, Heidelberg (2008)
- Endo, G., Morimoto, J., Matsubara, T., Nakanishi, J., Cheng, G.: Learning CPG-based biped locomotion with a policy gradient method: Application to a humanoid robot. *I. J. Robotic Res.* 27(2), 213–228 (2008)

- Erden, M.S., Leblebicioaglu, K.: Free gait generation with reinforcement learning for a six-legged robot. *Robot. Auton. Syst.* 56(3), 199–212 (2008)
- Fagg, A.H., Lotspeich, D.L., Hoff, J., Bekey, G.A.: Rapid reinforcement learning for reactive control policy design for autonomous robots. In: *Artificial Life in Robotics* (1998)
- Gaskett, C., Fletcher, L., Zelinsky, A.: Reinforcement learning for a vision based mobile robot. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS (2000)
- Geng, T., Porr, B., Wörgötter, F.: Fast biped walking with a reflexive controller and real-time policy searching. In: *Advances in Neural Information Processing Systems*, NIPS (2006)
- Glynn, P.: Likelihood ratio gradient estimation: an overview. In: *Winter Simulation Conference*, WSC (1987)
- Goldberg, D.E.: *Genetic algorithms*. Addison Wesley (1989)
- Gräve, K., Stückler, J., Behnke, S.: Learning motion skills from expert demonstrations and own experience using gaussian process regression. In: *Joint International Symposium on Robotics (ISR) and German Conference on Robotics*, ROBOTIK (2010)
- Guenter, F., Hersch, M., Calinon, S., Billard, A.: Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics* 21(13), 1521–1544 (2007)
- Gullapalli, V., Franklin, J., Benbrahim, H.: Acquiring robot skills via reinforcement learning. *IEEE on Control Systems Magazine* 14(1), 13–24 (1994)
- Hafner, R., Riedmiller, M.: Reinforcement learning on a omnidirectional mobile robot. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS (2003)
- Hafner, R., Riedmiller, M.: Neural reinforcement learning controllers for a real robot application. In: *IEEE International Conference on Robotics and Automation*, ICRA (2007)
- Hailu, G., Sommer, G.: Integrating symbolic knowledge in reinforcement learning. In: *IEEE International Conference on Systems, Man and Cybernetics* (SMC) (1998)
- Hester, T., Quinlan, M., Stone, P.: Generalized model learning for reinforcement learning on a humanoid robot. In: *IEEE International Conference on Robotics and Automation*, ICRA (2010)
- Huang, X., Weng, J.: Novelty and reinforcement learning in the value system of developmental robots. In: *Lund University Cognitive Studies* (2002)
- Ijspeert, A.J., Nakanishi, J., Schaal, S.: Learning attractor landscapes for learning motor primitives. in: *Advances in Neural Information Processing Systems*, NIPS (2003)
- Ilg, W., Albiez, J., Jedele, H., Berns, K., Dillmann, R.: Adaptive periodic movement control for the four legged walking machine BISAM. In: *IEEE International Conference on Robotics and Automation*, ICRA (1999)
- Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4, 237–285 (1996)
- Kalmár, Z., Szepesvári, C., Lörincz, A.: Modular Reinforcement Learning: An Application to a Real Robot Task. In: Birk, A., Demiris, J. (eds.) *EWLR 1997. LNCS (LNAI)*, vol. 1545, pp. 29–45. Springer, Heidelberg (1998)
- Kappen, H.: Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics: Theory and Experiment* 11 (2005)
- Katz, D., Pyuro, Y., Brock, O.: Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. In: *Robotics: Science and Systems*, R:SS (2008)
- Kimura, H., Yamashita, T., Kobayashi, S.: Reinforcement learning of walking behavior for a four-legged robot. In: *IEEE Conference on Decision and Control* (CDC) (2001)
- Kirchner, F.: Q-learning of complex behaviours on a six-legged walking machine. In: *EUROMICRO Workshop on Advanced Mobile Robots* (1997)

- Kirk, D.E.: Optimal control theory. Prentice-Hall, Englewood Cliffs (1970)
- Ko, J., Klein, D.J., Fox, D., Hähnel, D.: Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In: IEEE International Conference on Robotics and Automation (ICRA) (2007)
- Kober, J., Peters, J.: Policy search for motor primitives in robotics. In: Advances in Neural Information Processing Systems, NIPS (2009)
- Kober, J., Peters, J.: Policy search for motor primitives in robotics. Machine Learning Online First (2010)
- Kober, J., Mohler, B., Peters, J.: Learning perceptual coupling for motor primitives. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2008)
- Kober, J., Oztop, E., Peters, J.: Reinforcement learning to adjust robot movements to new situations. In: Robotics: Science and Systems Conference (R:SS) (2010)
- Kohl, N., Stone, P.: Policy gradient reinforcement learning for fast quadrupedal locomotion. In: IEEE International Conference on Robotics and Automation (ICRA) (2004)
- Kolter, J.Z., Ng, A.Y.: Policy search via the signed derivative. In: Robotics: Science and Systems (R:SS) (2009)
- Kolter, J.Z., Abbeel, P., Ng, A.Y.: Hierarchical apprenticeship learning with application to quadruped locomotion. In: Advances in Neural Information Processing Systems (NIPS) (2007)
- Kolter, J.Z., Coates, A., Ng, A.Y., Gu, Y., DuHadway, C.: Space-indexed dynamic programming: learning to follow trajectories. In: International Conference on Machine Learning (ICML) (2008)
- Kolter, J.Z., Plagmann, C., Jackson, D.T., Ng, A.Y., Thrun, S.: A probabilistic approach to mixed open-loop and closed-loop control, with application to extreme autonomous driving. In: IEEE International Conference on Robotics and Automation (ICRA) (2010)
- Kroemer, O., Detry, R., Piater, J., Peters, J.: Active learning using mean shift optimization for robot grasping. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2009)
- Kroemer, O., Detry, R., Piater, J., Peters, J.: Combining active learning and reactive control for robot grasping. Robotics and Autonomous Systems 58(9), 1105–1116 (2010)
- Kuhn, H.W., Tucker, A.W.: Nonlinear programming. In: Berkeley Symposium on Mathematical Statistics and Probability (1950)
- Latzke, T., Behnke, S., Bennewitz, M.: Imitative Reinforcement Learning for Soccer Playing Robots. In: Lakemeyer, G., Sklar, E., Sorrenti, D.G., Takahashi, T. (eds.) RoboCup 2006. LNCS (LNAI), vol. 4434, pp. 47–58. Springer, Heidelberg (2007)
- Lizotte, D., Wang, T., Bowling, M., Schuurmans, D.: Automatic gait optimization with gaussian process regression. In: International Joint Conference on Artificial Intelligence (IJCAI) (2007)
- Mahadevan, S., Connell, J.: Automatic programming of behavior-based robots using reinforcement learning. Artificial Intelligence 55(2-3), 311–365 (1992)
- Martínez-Marín, T., Duckett, T.: Fast reinforcement learning for vision-guided mobile robots. In: IEEE International Conference on Robotics and Automation (ICRA) (2005)
- Mataric, M.J.: Reward functions for accelerated learning. In: International Conference on Machine Learning (ICML) (1994)
- Mataric, M.J.: Reinforcement learning in the multi-robot domain. Autonomous Robots 4, 73–83 (1997)
- Michels, J., Saxena, A., Ng, A.Y.: High speed obstacle avoidance using monocular vision and reinforcement learning. In: International Conference on Machine Learning (ICML) (2005)

- Mitsunaga, N., Smith, C., Kanda, T., Ishiguro, H., Hagita, N.: Robot behavior adaptation for human-robot interaction based on policy gradient reinforcement learning. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2005)
- Miyamoto, H., Schaal, S., Gandolfo, F., Gomi, H., Koike, Y., Osu, R., Nakano, E., Wada, Y., Kawato, M.: A kendama learning robot based on bi-directional theory. *Neural Networks* 9(8), 1281–1302 (1996)
- Morimoto, J., Doya, K.: Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems* 36(1), 37–51 (2001)
- Nakanishi, J., Cory, R., Mistry, M., Peters, J., Schaal, S.: Operational space control: a theoretical and empirical comparison. *International Journal of Robotics Research* 27, 737–757 (2008)
- Nemec, B., Tamošiūnaitė, M., Wörgötter, F., Ude, A.: Task adaptation through exploration and action sequencing. In: IEEE-RAS International Conference on Humanoid Robots, Humanoids (2009)
- Nemec, B., Zorko, M., Zlajpah, L.: Learning of a ball-in-a-cup playing robot. In: International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD) (2010)
- Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E.: Autonomous inverted helicopter flight via reinforcement learning. In: International Symposium on Experimental Robotics (ISER) (2004a)
- Ng, A.Y., Kim, H.J., Jordan, M.I., Sastry, S.: Autonomous helicopter flight via reinforcement learning. In: Advances in Neural Information Processing Systems (NIPS) (2004b)
- Oßwald, S., Hornung, A., Bennewitz, M.: Learning reliable and efficient navigation with a humanoid. In: IEEE International Conference on Robotics and Automation (ICRA) (2010)
- Paletta, L., Fritz, G., Kintzler, F., Irran, J., Dorffner, G.: Perception and Developmental Learning of Affordances in Autonomous Robots. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) KI 2007. LNCS (LNAI), vol. 4667, pp. 235–250. Springer, Heidelberg (2007)
- Pastor, P., Kalakrishnan, M., Chitta, S., Theodorou, E., Schaal, S.: Skill learning and task outcome prediction for manipulation. In: IEEE International Conference on Robotics and Automation (ICRA) (2011)
- Pendrith, M.: Reinforcement learning in situated agents: Some theoretical problems and practical solutions. In: European Workshop on Learning Robots (EWRL) (1999)
- Peters, J., Schaal, S.: Learning to control in operational space. *International Journal of Robotics Research* 27(2), 197–212 (2008a)
- Peters, J., Schaal, S.: Natural actor-critic. *Neurocomputing* 71(7-9), 1180–1190 (2008b)
- Peters, J., Schaal, S.: Reinforcement learning of motor skills with policy gradients. *Neural Networks* 21(4), 682–697 (2008c)
- Peters, J., Vijayakumar, S., Schaal, S.: Linear quadratic regulation as benchmark for policy gradient methods. Tech. rep., University of Southern California (2004)
- Peters, J., Mülling, K., Altun, Y.: Relative entropy policy search. In: National Conference on Artificial Intelligence (AAAI) (2010a)
- Peters, J., Mülling, K., Kober, J., Nguyen-Tuong, D., Kroemer, O.: Towards motor skill learning for robotics. In: International Symposium on Robotics Research, ISRR (2010b)
- Piater, J., Jodogne, S., Detry, R., Kraft, D., Krüger, N., Kroemer, O., Peters, J.: Learning visual representations for perception-action systems. *International Journal of Robotics Research Online First* (2010)
- Platt, R., Grupen, R.A., Fagg, A.H.: Improving grasp skills using schema structured learning. In: International Conference on Development and Learning (2006)
- rAström, K.J., Wittenmark, B.: Adaptive control. Addison-Wesley, Reading (1989)

- Riedmiller, M., Gabel, T., Hafner, R., Lange, S.: Reinforcement learning for robot soccer. *Autonomous Robots* 27(1), 55–73 (2009)
- Rottmann, A., Plagemann, C., Hilgers, P., Burgard, W.: Autonomous blimp control using model-free reinforcement learning in a continuous state and action space. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2007)
- Rückstieß, T., Felder, M., Schmidhuber, J.: State-Dependent Exploration for Policy Gradient Methods. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part II. LNCS (LNAI), vol. 5212, pp. 234–249. Springer, Heidelberg (2008)
- Sato, M.-A., Nakamura, Y., Ishii, S.: Reinforcement Learning for Biped Locomotion. In: Dorronsoro, J.R. (ed.) ICANN 2002. LNCS, vol. 2415, pp. 777–782. Springer, Heidelberg (2002)
- Schaal, S.: Learning from demonstration. In: Advances in Neural Information Processing Systems, NIPS (1997)
- Schaal, S., Atkeson, C.G.: Robot juggling: An implementation of memory-based learning. *Control Systems Magazine* 14(1), 57–71 (1994)
- Schaal, S., Atkeson, C.G., Vijayakumar, S.: Scalable techniques from nonparametric statistics for real-time robot learning. *Applied Intelligence* 17(1), 49–60 (2002)
- Schaal, S., Mohajerian, P., Ijspeert, A.J.: Dynamics systems vs. optimal control - a unifying view. *Progress in Brain Research* 165(1), 425–445 (2007)
- Smart, W.D., Kaelbling, L.P.: A framework for reinforcement learning on real robots. In: National Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, AAAI/IAAI (1998)
- Smart, W.D., Kaelbling, L.P.: Effective reinforcement learning for mobile robots. In: IEEE International Conference on Robotics and Automation (ICRA) (2002)
- Soni, V., Singh, S.: Reinforcement learning of hierarchical skills on the sony aibo robot. In: International Conference on Development and Learning (ICDL) (2006)
- Strens, M., Moore, A.: Direct policy search using paired statistical tests. In: International Conference on Machine Learning (ICML) (2001)
- Sutton, R., Barto, A.: Reinforcement Learning. MIT Press, Boston (1998)
- Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: International Machine Learning Conference (1990)
- Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems (NIPS) (2000)
- Sutton, R.S., Koop, A., Silver, D.: On the role of tracking in stationary environments. In: International Conference on Machine Learning (ICML) (2007)
- Svinin, M.M., Yamada, K., Ueda, K.: Emergent synthesis of motion patterns for locomotion robots. *Artificial Intelligence in Engineering* 15(4), 353–363 (2001)
- Tamei, T., Shibata, T.: Policy Gradient Learning of Cooperative Interaction with a Robot Using User's Biological Signals. In: Köppen, M., Kasabov, N., Coghill, G. (eds.) ICONIP 2008. LNCS, vol. 5507, pp. 1029–1037. Springer, Heidelberg (2009)
- Tedrake, R.: Stochastic policy gradient reinforcement learning on a simple 3d biped. In: International Conference on Intelligent Robots and Systems (IROS) (2004)
- Tedrake, R., Zhang, T.W., Seung, H.S.: Learning to walk in 20 minutes. In: Yale Workshop on Adaptive and Learning Systems (2005)
- Tedrake, R., Manchester, I.R., Tobenkin, M.M., Roberts, J.W.: LQR-trees: Feedback motion planning via sums of squares verification. *International Journal of Robotics Research* 29, 1038–1052 (2010)

- Theodorou, E.A., Buchli, J., Schaal, S.: Reinforcement learning of motor skills in high dimensions: A path integral approach. In: IEEE International Conference on Robotics and Automation (ICRA) (2010)
- Thrun, S.: An approach to learning mobile robot navigation. *Robotics and Autonomous Systems* 15, 301–319 (1995)
- Tokic, M., Ertel, W., Fessler, J.: The crawler, a class room demonstrator for reinforcement learning. In: International Florida Artificial Intelligence Research Society Conference (FLAIRS) (2009)
- Toussaint, M., Storkey, A., Harmeling, S.: Expectation-Maximization methods for solving (PO)MDPs and optimal control problems. In: *Inference and Learning in Dynamic Models*. Cambridge University Press (2010)
- Touzet, C.: Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems, Special Issue on Learning Robot: the New Wave* 22(3-4), 251–281 (1997)
- Uchibe, E., Asada, M., Hosoda, K.: Cooperative behavior acquisition in multi mobile robots environment by reinforcement learning based on state vector estimation. In: IEEE International Conference on Robotics and Automation (ICRA) (1998)
- Vlassis, N., Toussaint, M., Kontes, G., Piperidis, S.: Learning model-free robot control by a Monte Carlo EM algorithm. *Autonomous Robots* 27(2), 123–130 (2009)
- Wang, B., Li, J., Liu, H.: A heuristic reinforcement learning for robot approaching objects. In: IEEE Conference on Robotics, Automation and Mechatronics (2006)
- Willgoss, R.A., Iqbal, J.: Reinforcement learning of behaviors in mobile robots using noisy infrared sensing. In: Australian Conference on Robotics and Automation (1999)
- Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 229–256 (1992)
- Yasuda, T., Ohkura, K.: A Reinforcement Learning Technique with an Adaptive Action Generator for a Multi-Robot System. In: Asada, M., Hallam, J.C.T., Meyer, J.-A., Tani, J. (eds.) *SAB 2008. LNCS (LNAI)*, vol. 5040, pp. 250–259. Springer, Heidelberg (2008)
- Youssef, S.M.: Neuro-based learning of mobile robots with evolutionary path planning. In: ICGST International Conference on Automation, Robotics and Autonomous Systems (ARAS) (2005)

Part VI

Closing

Chapter 19

Conclusions, Future Directions and Outlook

Marco Wiering and Martijn van Otterlo

19.1 Looking Back

This book has provided the reader with a thorough description of the field of reinforcement learning (RL). In this last chapter we will first discuss what has been accomplished with this book, followed by a description of those topics that were left out of this book, mainly because they are outside of the main field of RL or they are small (possibly novel and emerging) subfields within RL. After looking back what has been done in RL and in this book, a step into the future development of the field will be taken, and we will end with the opinions of some of the authors what they think will become the most important areas of research in RL.

19.1.1 *What Has Been Accomplished?*

The book has the aim to describe a very large part of the field of RL. It focuses on all big directions of RL that were not covered in the important books on RL by Sutton et al (1998) and by Bertsekas and Tsitsiklis (1996). Starting from the most well known methods in RL described in chapter 2, several chapters are devoted to describing more efficient solution methods than those proposed more than 15 years ago. The large body of work and the number of citations in this book clearly show the enormous growth of the field since the initial contributions, most importantly probably the TD algorithm (Sutton, 1988) and the discovery of the Q-learning

Marco Wiering
Department of Artificial Intelligence, University of Groningen, The Netherlands
e-mail: m.a.wiering@rug.nl

Martijn van Otterlo
Radboud University Nijmegen, The Netherlands
e-mail: m.vanotterlo@donders.ru.nl

algorithm (Watkins, 1989; Watkins and Dayan, 1992). Many of these newer developments make better use of an agent's experiences, such as batch RL (chapter 2), least squares methods for policy iteration (chapter 3), and the use of models (chapter 4) and knowledge transfer (chapter 5). Chapter 6 analyzes theoretical advantages of better exploration methods to obtain more important experiences.

In the third part of the book different usages of a variety of representations in RL are given. Such representations can be vector based as in chapter 7, use first-order logic (chapter 8), make efficient use of hierarchies (chapter 9), or are basically representation bias free as when evolutionary algorithms are used (chapter 10). When the right representation is used, learning can be made much more effective and the learned policies can be sometimes much more intelligible as in the case of first-order logic programs and value functions.

In the fourth part different probabilistic frameworks and algorithms are described. In chapter 11 the novel framework of Bayesian RL is described. In chapter 12, partially observable Markov decision processes and efficient solution methods are covered. Chapter 13 describes predictive state representations where agent histories are compactly described by a set of expectancies about the future. In chapter 14, an extension to multiple agents is given together with game theoretical notions of cooperation, coordination, and competition. This chapter is followed by a description of the decentralized partially observable Markov decision process framework and planning algorithms for solving these hard problems, in chapter 15. The book ends with background information and the relation to human learning in chapter 16, successful applications of RL for learning games in chapter 17, and using RL methods for robot control in chapter 18.

19.1.2 Which Topics Were Not Included?

RL is a topic within the field of machine learning. Very often unsupervised and supervised machine learning methods are used for applying RL effectively in large state-action spaces. This book has the goal to describe recent developments in RL, and therefore machine learning in general is not well covered. Of course reinforcement learning techniques can be combined with many kinds of regression methods to learn the value functions from a limited amount of interactions with the environment. Therefore, new developments in learning regression models are very important as well for the field of RL. We refer to some books, see, e.g. (Mitchell, 1996; Alpaydin, 2010; Bishop, 2006), on machine learning to allow the reader to study the whole field of machine learning, although most of the existing machine learning books cover RL in a single chapter only.

Next to being related to machine learning, RL shares the same objectives as the use of planning algorithms or control theory. Often these fields use a slightly different notation, but the shared objective is to allow an agent or controller to select actions so that its design objectives are optimally fulfilled. We cannot cover these

fields next to the field of RL, because they have a much longer background and a larger community.

Therefore, we will discuss topics that are not included which are commonly referred to as RL techniques, although they do not all need to learn value functions such as also discussed in the chapter on evolutionary RL (Moriarty and Miikkulainen, 1996). We will first discuss some areas in RL that were not covered in this book, although they show interesting new developments in this rich field. After that we will discuss some application areas that have shown effective usage of RL for solving difficult problems.

19.1.2.1 More Developments in Reinforcement Learning

Of course a book can never describe an entire field, especially not if the field exists for more than two decades, and when the research community is growing every year. Therefore, the following list can never be complete, but it shows some interesting ideas that can prove useful for more efficient RL for solving particular problems.

Online planning and roll-out methods. For large state-action spaces it can be very hard to accurately approximate the value function with a limited amount of resources. Often the function approximator cannot perfectly fit the target value function, and this will cause the selection and execution of wrong actions. One partial remedy for this problem is to use look-ahead planning in order to use more information in the decision making process. This technique was for example applied successfully to make TD-Gammon (Tesauro, 1995) even stronger as a backgammon playing program. Another interesting technique that receives a lot of attention in the community of applying methods from artificial and computational intelligence for game playing programs, is the Monte Carlo Tree Search (MCTS) method. This method is based on running many simulations in a quick way to sample the results of many future paths of the agent. MCTS has been very effective for playing the game of Go (Coulom, 2006). Such methods try to avoid the immediate use of a learned value function that is not an accurate approximation to the optimal value function. The idea is that these techniques can be more robust against uncertainties in estimates by averaging the outcomes over many simulations that generate different future behaviors. Another technique that falls in the area of online planning methods is model-based predictive control (MPC). This technique is well known in the control theory field and uses a moving horizon window to compute the return of different limited action sequences. The “plan” that obtains most reward will then be used to execute the first action, after which the planning process is repeated. These MPC techniques can be very useful for deterministic tasks, but may have problems with dealing with many uncertainties or stochastic outcomes.

Curiosity and internal motivations. Agents can profit from learning a sequence of subtasks such as sitting down before starting to use the computer. Learning adaptive subgoals sometimes relies on the use of highly frequent state visits to identify bottleneck states, but such methods need obviously a large amount of experiences. Internal motivations and the use of creativity to maximize some generated reward

function, either specified a-priori or not, can give RL and possibly model-based RL in particular, a boost for understanding the world. One development (Schmidhuber, 1991b,a, 2010) is the use of curiosity and creativity for generating trivial, novel, and surprising behaviors and data. Such algorithms should have two learning components: a general reward optimizer or reinforcement learner, and an adaptive encoder of the agent's growing data history (the record of the agent's interaction with its environment). The learning progress of the encoder is the intrinsic reward for the reward optimizer. That is, the latter is motivated to invent *interesting* spatio-temporal patterns that the encoder does not yet know but can easily learn to encode better with little computational effort. To maximize expected reward (in the absence of external reward), the reward optimizer will create more and more-complex behaviors that yield temporarily surprising (but eventually boring) patterns that make the encoder quickly improve.

Model-based RL with adaptive state partitioning. Model-based algorithms first estimate a model based on the experiences of an agent, and then use dynamic programming methods to compute a policy. The advantage is that once the transition model is estimated, it is possible to use new reward functions, e.g. one that gives a reward of one for entering a goal state and zero elsewhere, and a new policy can be computed. Such knowledge reuse is often not possible with model-free RL methods. Although model-based techniques can be very sample efficient, they have problems with dealing with continuous states and actions, since it is hard to store the transition probabilities (or probability densities) in this case. One possible direction to cope with this is to use adaptive state partitioning to cluster similar states in an abstract state, and to estimate transition probabilities between abstract states given some discrete action. This approach may lead to models which do not obey the Markov property anymore, and therefore it can make sense to combine them with memory of previous states and actions, as is done in several techniques for solving POMDPs (Ring, 1994; McCallum, 1995).

Other RL methods and function approximation techniques. There is a large number of RL methods proposed in literature. E.g. one paper (Wiering and van Hasselt, 2009) proposed five new RL methods. Other recent methods are double Q-learning (van Hasselt, 2010) and Speedy Q-learning (Azar et al, 2011). Often these algorithms have some advantages in particular environments, but may perform worse in other environments. Since the list of RL algorithms is so large, this book has not covered all of them. The same holds for RL algorithms in combination with a variety of function approximators. We did not extensively cover the use of recurrent neural networks in combination with RL, see e.g. (Schmidhuber, 1990; Bakker and Schmidhuber, 2004), although they can be important tools for POMDPs or multi-agent environments. In multi-agent RL it can be very beneficial to memorize previous interactions in order to model other agents, and possibly lead them to behaviors which are advantageous for the agent itself. Other function approximators, such as support-vector machines (Vapnik, 1995) have been used in RL (Dietterich and Wang, 2002), but have not received a large interest from the

RL community, since RL researchers often use generalized linear models or neural networks instead.

Safe exploration and issues for real-time learning. When RL is applied directly on robots or for controlling other physical devices, it would be very important that the system is not damaged by the adaptive controllers that often learn without any form of a-priori knowledge, i.e., from Tabula Rasa. Also in the case of training RL agents that interact with human users, it is not desirable that the RL agent explores many actions so that the human user can be frustrated by the stupid actions of the agent (Westra, 2011). Therefore, in such cases it may be necessary to combine RL with pre-existing, safe, controllers. RL can then be used to change the output of the controller in many circumstances, but the controller may also prevent the RL agent from changing its generated actions.

Another interesting issue for real-time learning is the use of a-priori knowledge. Many RL researchers have shown that RL can learn without the use of a-priori knowledge. This, however, is often at the cost of a large number of interactions that are needed before the agent performs reasonably well. Instead of learning from scratch, some research has relied on presenting the agent a-priori knowledge, possibly in the form of an existing controller or generating initial behaviors by showing the agent demonstrations (e.g. (Smart and Kaelbling, 2002; Coates et al, 2009)). It may also be possible to give an abstract representation of the task (van Otterlo, 2003) so that only the Q-values or other parameters need to be learned. It is important to use a-priori knowledge, if such knowledge is easily available and difficult to learn by the agent.

Program evolution. Several methods have been proposed that share the goal of RL, but reach it in quite different ways. One of these methods are classifier systems, which originally used the bucket brigade algorithm to assign credit and blame to condition-action rules that have been used by the agent in its past. Classifier systems are also used in combination with evolutionary algorithms to evolve a good set of rules, and possibly for this reason they are more studied in the field of evolutionary computation. EIRA (Schmidhuber, 1996) is a principle that allows an agent to change its own program and keep the changes if they lead to more cumulative rewards obtained from the environment. EIRA has been used in combination with Levin Search (Wiering and Schmidhuber, 1996; Schmidhuber et al, 1997c) and has led to the success-story algorithm (Schmidhuber et al, 1997b,a,c), an algorithm that uses self-modifying policies that provably lead to higher and higher reward intakes. Genetic programming (Cramer, 1985; Koza, 1992, 1994) and related techniques such as PIPE (Sałustowicz and Schmidhuber, 1997) have also been used for solving RL problems. Like classifier systems, they are more studied in the evolutionary computation community.

Universal agents and optimal problem solving. There is also a growing body of papers which is researching theoretical optimal algorithms for solving control problems, which is another interesting development, although this development is based on quite different principles. The theory of Kolmogorov complexity (Kolmogorov, 1965; Solomonoff, 1964; Li and Vitányi, 1990) provides a basis for

optimal prediction and universal inductive inference (Solomonoff, 1964, 1978). The theoretically optimal yet uncomputable RL algorithm Aixi (Hutter, 2004) has found several implementations now (Poland and Hutter, 2006; Veness et al, 2011), an example of which is playing PacMan.

Most of these theoretical algorithms are based on finding the shortest program that solves a problem. A problem of this is that often many programs need to be tested, in order of complexity. In case a short program is able to compute the optimal solution, then this approach can be very efficient. Some developments in this line of research have taken place. First, the Speed Prior is derived from the fastest (not the shortest) way of computing data, and it was shown that it allows for deriving a *computable* strategy for optimal prediction of the future given the past (Schmidhuber, 2002). Unlike Levin search, the algorithm called optimal ordered problem solving (OOPS) does not ignore the huge constants buried in the algorithmic complexity. OOPS incrementally reduces the constants before the $O()$ notation through experience (Schmidhuber, 2004). Within a few days OOPS learned a universal solver for all n disk *Towers of Hanoi* problems, solving all instances up to $n = 30$, where the shortest solution (not the search for it!) takes more than 10^9 moves. Gödel machines (Schmidhuber, 2009) are the first class of mathematically rigorous, general, fully self-referential, self-improving, optimally efficient RL machines or problem solvers. Inspired by Gödel's celebrated self-referential formulas (1931), a Gödel machine rewrites any part of its own code as soon as it has found a proof that the rewrite is useful.

19.1.2.2 Different Application Areas of RL Methods

Although this book has described two application areas, namely robotics and game playing, there has also been quite some research in using RL for letting agents learn to act in several other interesting areas. Some of these areas will be described below.

Economical applications. In economical applications the aim of an agent is to earn as much money as possible in a reasonable time. One example is the use of RL agents to trade stocks in stock markets or to sell and buy foreign currencies given fluctuating exchange rates (Nevmyvaka et al, 2006). In such problems it is very difficult or even impossible to predict the future. Such techniques often rely on letting agents learn to optimize their gains given the past dynamics of the prices and after that use the best agent to trade in the near future. Although this approach may work, it is often very difficult to deal with unpredictable events such as stock market crashes. Another economical application is to have agents trade objects on the internet or to organize trips for a booking company. Such trading agents may have many benefits, such as their speed and low-cost usage, compared to human traders and organizers.

Network applications. Many real-world problems can be modelled with networks where some items are travelling over the edges. One of the first successful applications of RL in the area of optimizing agents in a network of nodes was routing messages on the internet (Littman and Boyan, 1993; Di Caro and Dorigo, 1998). In

such networks messages arrive at nodes and an agent in a node learns the optimal way for directing the message to its goal-node. Especially in quite saturated networks, these adaptive methods are very promising. Similar to routing messages on the internet, RL agents can be used in smart-grid applications in order to control the energy flow and storage on an electricity grid. Another type of network can be found in daily traffic, where vehicles have a particular destination, and at intersections (the nodes), traffic lights are operational to circumvent collisions. In (Wiering, 2000), RL was used to optimize the agents that control traffic lights in such a network, and was shown to outperform a variety of fixed controllers. As a last application, we mention the use of RL agents to optimize the use of cognitive radio and mobile phones frequency channels, where the RL agents have the goal to minimize lost calls that require a particular channel frequency.

Learning to communicate. RL agents can also be used to select actions related to natural language. Saying or writing some words can be considered to be a particular type of action. The difficulty is that there are a huge number of short natural language messages possible due to the huge amount of existing words. To deal with this issue, one interesting paper (Singh et al, 2002) describes the use of a fixed number of sentences that can be told to people phoning a computer to inquire about particular upcoming events. Because the computer uses speech recognition to understand the human user, it cannot always be certain about the information a caller wants to have. Therefore, the probabilistic framework of MDPs and POMDPs can be fruitful for such tasks.

Combinatorial optimization problems. The field of operations research and meta-heuristics is very large, and has the aim to solve very complex problems that do not necessarily require interaction with an agent. Often these problems are formulated with matrices describing the problem state and a cost function stating which solution is optimal. One type of combinatorial optimization solving method, called ant colony optimization (Dorigo et al, 1996) uses ants to make local decisions, similar to RL agents. After each ant has performed a sequence of local decisions, a solution is obtained. After that the best sequence of decisions is rewarded with additional pheromone, which influences future decisions of all ants. In this way, ant colony optimization is very similar to RL, and it has been applied successfully to problems such as the traveling salesman problem (Dorigo and Gambardella, 1997), the quadratic assignment problem (Gambardella et al, 1999) and internet traffic routing (Di Caro and Dorigo, 1998). Also other multi-agent RL methods have been used for combinatorial optimization problems, such as for job-shop scheduling and load balancing (Riedmiller and Riedmiller, 1999).

Experimental testbeds. When researchers develop a new RL method, they often perform experiments with it to show how well the new algorithm is performing compared to previous methods. A problem when one compares the obtained results of different algorithms described in different papers is that often the used problem parameters are slightly different. E.g., the mountain car problem becomes much simpler to solve and requires fewer actions to climb the hill, if the discrete time-step is a bit larger. For this reason, some researchers have developed RL Glue

(Tanner and White, 2009) with which competitions have been held between RL algorithms. RL Competitions have been held for challenging domains such as Tetris and Mario Bros. Such experimental testbeds allow for fair comparisons between different methods, and such standardized platforms should help to answer the question: “which RL method performs best for which environment?”.

19.2 Looking into the Future

RL techniques have a lot of appeal, since they can be applied to many different problems where an agent is used to perform tasks autonomously. Although RL has already been used with success for many toy and some real-world problems, a lot of future work is still needed to apply RL for many problems of interest. This section will first look at research questions that have not been fully answered yet, and then examines some applications that seem unsolvable with RL methods.

19.2.1 *Things That Are Not Yet Known*

We will first describe research questions in RL, which have so far remained unanswered, although the field of RL could become even more efficient for solving many kinds of problems when knowing the answers.

What is the best function approximator? RL algorithms have often been combined with tabular representations for storing the value functions. Although most of these algorithms have been proven to converge, for scaling up to large or continuous state-action spaces, function approximators are necessary. For this reason most RL researchers use generalized linear models where a fixed set of basis functions is chosen, or multi-layer perceptrons. The advantage of linear models where only the parameters between basis function activations and the state value estimates have to be learned, is that they are fast and allow for techniques from linear algebra to efficiently compute the approximations. A disadvantage is that this method requires a set of basis functions which has to be chosen a-priori. Furthermore, often the used basis functions are very local, such as fuzzy rules or radial basis functions, and these tend not to scale up well to problems with very many input dimensions. Neural networks can learn the hidden features and work well with logistic activation functions that create shorter hidden representations than more localized basis functions. Furthermore, they are more flexible since they can learn the placement of the basis functions, but training them with RL often takes a lot of experiences. For representing the value functions, basically all regression algorithms from machine learning can be used. For example, in (Dietterich and Wang, 2002), support vector machines were proposed in combination with RL and in (Ernst et al, 2005) random forests were used. Recently several feature construction techniques – e.g. based on reward based heuristics or Bellman residuals – are being developed for RL and dynamic programming, but much work is still needed to find out when and how

to apply them in the most effective way. In the book of Sutton and Barto (1998), Kanerva encoding is proposed as a representation that could possibly scale up to many input dimensions. The question that still remains to be solved is for what type of problem which function approximator works best.

Convergence proofs for general algorithms. There has been quite some research to study convergence of RL algorithms in combination with particular function approximators, e.g., with generalized linear models. Some research in this direction (Schoknecht, 2002; Maei et al, 2010) has shown that some combinations can be proven to converge given particular assumptions. Often these proofs do not extend to the full control case, but only apply to the evaluation case where the value function of a fixed policy is estimated. We would like to see more proofs that guarantee the convergence of many function approximators, and what the conditions of these proofs would be. Furthermore, the rate of convergence also plays a very important role. Such research could help practitioners to choose the best method for their specific problem.

Optimal exploration. A lot of research has focused on optimal exploration to find the most important learning experiences. However, it is still the question which exploration method will work best for particular problems. The problem of choosing the optimal exploration policy for a problem becomes also much harder when function approximators are used instead of lookup tables. Some research (Nouri and Littman, 2010) has proposed an efficient exploration method for continuous spaces. Such problems are very hard, since given a limited training time of the agent, the agent is often not able to visit all parts of the environment. Therefore trade-offs need to be made. Creating the most efficient exploration strategy for difficult problems is still an underdeveloped topic in RL, and therefore most researchers are still using the simple ε -greedy and Boltzmann exploration for their problems at hand.

Scaling up issues. RL has already been applied successfully to particular complex problems involving many input dimensions. The most famous examples are TD-Gammon (Tesauro, 1995) and Elevator Dispatching (Crites and Barto, 1996). However, fifteen years later, still little is known about what is required to let RL scale-up to solve basically any control problem. It is quite probable that RL will perform very well for some problems and much worse for other problems. For example, the success of TD-Gammon can be partially explained by the smooth value functions caused by the stochasticity of the Backgammon game. It is well known in machine learning that approximating smooth functions is much simpler than approximating very fluctuating, or volatile, functions. It is important to understand more about the problems faced when applying RL to solve complex problems, since this will allow the field to be applied to solve many industrial applications, for which currently often other methods, e.g. from control theory, are applied.

How does the brain do RL? The old brain contains areas that generate emotions and feelings and is connected to perceptual input. These emotions play a very important role from the beginning of a children's life and can be translated in RL terms as providing rewards to the cortex. There are also feedback loops in humans from

the frontal cortex to the old brain, so that a person can think and thereby manipulate the way (s)he feels. Although a rough analogy can be made between RL systems and the brain, we still do not understand enough about the exact workings of the brain, which would allow for programming more human-like learning abilities in agents. A lot of research in neuroscience is focusing on studying detailed workings of the brain and measurement devices become more and more accurate, but we expect that there is still a long way to go until neuroscientists understand all human learning processes in a detailed way.

19.2.2 Seemingly Impossible Applications for RL

We have the feeling that RL can be used for many different control tasks, which require an agent that makes decisions to manipulate a given environment. Below, we give some examples of applications that seem too hard for RL to solve.

General intelligence only with RL. General intelligence requires perception, natural language processing, executing complex behaviors, and much more. For these different tasks, different subfields in artificial intelligence are focused at developing better and better algorithms over time. The field of artificial general intelligence tries to develop agents that possess all such different skills. It is not advisable to use only RL for creating such different intelligent skills, although RL could play a role in each of them. The ability to predict the future is maybe one of the most important abilities of intelligent entities. When we see some car driving over a road, we can predict where it will be after some seconds with high accuracy. When we walk down a staircase we predict when our feet will hit the stairs. RL can be fruitfully applied to solve prediction problems, and therefore it can play an important role in the creation of general intelligence, but it cannot do this without developments in other subfields of artificial intelligence.

Using value-function based RL to implement programs. One control problem that RL researchers often face is the task of implementing a program involving an RL agent and an environment. It would be very interesting to study RL methods that can program themselves, such as the Gödel machine (Schmidhuber, 2009). This could allow RL agents to program even better RL agents, which will then never stop. It is according to us very unlikely that current value-function based RL methods will be suited for this. The reason is that searching for the right program can be seen as finding a needle in a haystack problem: almost no programs do anything useful, and only few solve the task. Without being able to learn from partial rewards, an RL optimization method for creating programs is unlikely to work. We want to note that there has been an efficient application of RL to optimize compilers (McGovern and a. Andrew G. Barto, 1999), but that application of RL used a lot of domain knowledge to make it work well. The use of RL in general programming languages is a related, and interesting, direction for further research (Simpkins et al, 2008).

This would enable to write a program with all available knowledge, and let RL optimize all other aspects of that program.

19.2.3 *Interesting Directions*

We will now give some directions that could improve the efficiency of RL. Furthermore, we describe some application areas that do not fully use the potential of RL algorithms.

Better world modelling techniques. Model-based RL can use learning experiences much more effectively than model-free RL algorithms. Currently, many model building techniques exist, such as batch RL (Riedmiller, 2005; Ernst et al, 2005), LSPI (see chapter 3 in this book), Prioritized sweeping (Moore and Atkeson, 1993), Dyna (Sutton, 1990), and best match learning (van Seijen et al, 2011). The question is whether we can use vector quantization techniques to discretize continuous state spaces, and then apply dynamic programming like algorithms in promising novel ways. For example, for a robot navigation task, the famous SIFT feature (Lowe, 2004) can be used to create a set of visual keywords. Then multiple substates would become active in the model, which makes the planning and representation problem challenging. It would for example be possible to estimate the probability that visual keywords are activated at the next time step from the previous active keywords using particular regression algorithms, and combine active keywords to estimate reward and value functions. Such a research direction creates many new interesting research questions.

Highly parallel RL. With more and more distributed computing facilities that can be used, it is interesting to research RL methods that can optimally profit from recent developments in cloud computing and supercomputing. In case of linear models, research can be performed to see if averaging the weights from the linear models will create better models than the separate ones. Some research has also focused on ensemble methods in RL (Wiering and van Hasselt, 2008) and these methods can be advantageous when distributed computing facilities are used.

Combining RL with learning from demonstration. Often controllers or agents already exist for many applications, and this allows RL agents to first learn from demonstrated experiences, after which the RL agents can finetune themselves to optimize the performance even further. In robotics, learning from demonstration has already been applied successfully, see e.g. (Coates et al, 2009). However, in one application of learning from demonstration to learn the game of backgammon, no performance improvement was obtained (Wiering, 2010). A challenging research direction is to explore novel problems where learning from demonstration can be very successfully applied in combination with RL.

RL for learning to play very complex games. Some very complex games, such as Go, are currently best played by a computer that uses Monte-Carlo tree search (Coulom, 2006). RL methods can hardly be employed well for playing such games.

The reason is that it is very difficult to accurately represent and learn the value function. Therefore, a lot of search and sampling currently works better than knowledge intensive solutions. The search/knowledge tradeoff (Berliner, 1977) states that more knowledge will come at the expense of less search, since models containing more parameters consume more time to evaluate. It would be very interesting to gain understanding of novel RL methods that can quite accurately learn to approximate the value function for complex games, but which are still very fast to evaluate. Some research in this direction was performed for the game of chess (Baxter et al, 1997; Veness et al, 2009) where linear models trained with RL could be efficiently used by the search algorithms.

Novel queueing problems. Many successful RL applications, such as network routing (Littman and Boyan, 1993), elevator dispatching (Crites and Barto, 1996), and traffic light control (Wiering, 2000) are very related to queueing processes, where some items have to wait for others. An advantage of such problems is that often direct feedback is available. It would be interesting to study RL methods for novel queuing problems, such as train scheduling, where expected traveling times of many different individual trips are minimized by multi-agent RL techniques. Many research disciplines may profit from RL methods, such as telecommunications and traffic engineering, which can lead to a broader acceptance of RL as solution method.

19.2.4 Experts on Future Developments

Finally, to conclude the book, we asked the authors to comment on the following questions:

- Which direction(s) will it go in the future with RL?
- Which topics in RL do you find most important to be understood better and improved more?

The answers of some of the expert's "best educated guesses" are described below.

Ashvin Shah: Because of their focus on learning through interaction with the environment with limited prior knowledge and guidance, many reinforcement learning (RL) methods suffer from the curse of dimensionality. The inclusion of abstract representations of the environment and hierarchical representations of behavior allows RL agents to circumvent this curse to some degree. One important research topic is the autonomous development of useful abstractions and hierarchies so that reliance on prior knowledge and guidance can continue to be limited. Inspiration may be drawn from many sources, including studies on how animals develop and use such representations, and resulting research can produce more capable, flexible, and autonomous artificial agents.

Lucian Busoniu: Fueled by advances in approximation-based algorithms, the field of RL has greatly matured and diversified over the last few years. One issue that

still requires attention is the scarcity of convincing benchmarks, especially in the area of RL applied to physical systems, where RL is often still benchmarked on problems that are simple and/or solvable by more classical techniques, such as the car on the hill, inverted pendulum, or bicycle balancing. In this context of real-time control of physical systems, a fundamental requirement that has to be addressed is guaranteeing safety: the learning process must not damage the system or endanger its users. The interplay between this requirement and exploration is expected to play a key role here, as are control-theoretic insights such as those into the stability of systems.

Todd Hester: Some of the most important current and future research directions in RL are towards making it practical and effective on more complex and real-world problems. To this end, we believe that research in RL will focus on algorithms that are simultaneously sample efficient enough to work with limited data even in large, continuous state spaces *and* computationally efficient enough to work in real time. Model-based RL is an effective way to obtain low sample efficiency, and should continue to be a focus of future RL research. To improve computational efficiency of model-based algorithms in complex state spaces, one promising direction is to develop algorithms that utilize parallel processors available on many computers and robots.

Lihong Li: What we need is a mathematical theory on generalization for function approximation in RL. Important developments are the principled use of prior knowledge in RL with theoretical guarantees for exploration, model learning, value-function learning, etc. Finally, we need *Robust* off-policy RL from batch data that have small bias and variance. Here, an unbiased evaluation method will, on average, reveal the true total reward of an algorithm if it is run online.

Ann Nowé: As devices are getting more often interconnected in an ad hoc manner, multi-agent RL can be expected to become a major approach to organize those systems that are hard to model a priori. A major challenge will be to improve the current algorithms in terms of learning time, possibly by extending improvements from single agent RL to MAS. Moreover, most methods assume some particular setting, e.g. they assume all agents share a common interest, or the agents are involved in a social dilemma. As the environments become more complex, the agents might be involved in heterogeneous, unknown interactions. Therefore, more general methods will be required which can deal with a larger variety of situations.

Frans Oliehoek: Probably the biggest challenge in single-agent decision making is overcoming Bellman's curse of dimensionality. The number of states for realistic problems is huge (and often infinite). As such it is crucial to deepen our understanding of generalization and function approximation, especially in combination with feature selection methods. I think it will be necessary to abandon the notion of states and fundamentally focus on sub-state (feature-based) representations. The problem with the notion of state and the associated Markov property is that they trivialize the dynamic prediction problem. The curse of dimensionality is a direct consequence of this and I believe that it can be mitigated by explicitly learning,

representing and exploiting the influence of state variables, features and actions on each other. This may also lead to a better integration with ideas from the multi-agent planning community, where people have been exploiting limited influence of 'local states' (sub-sets of features) and actions on each other.

Jan Peters: RL is at a way-point where it essentially can go into two direction: First, we could consolidate classical RL and subsequently fall into obscurity. Second, we could return back to the basic questions and solve these with more solid answers as done in supervised learning, while working on better applications. I hope it will be the latter. Key steps for future RL will be to look more at the primal problem of RL (as done e.g., in (Peters et al, 2010)) and look how changes in the primal change the dual. Some RL methods are totally logical from this perspective – some will turn out to be theoretically improper. This may include classical assumptions on the cost functions of value function approximation such as TD or Bellman errors, as already indicated by Schoknecht's work Schoknecht (2002). The second component will be proper assumptions on the world. Human intelligence is not independent from the world we live in, neither will be robot RL. In robot reinforcement learning, I would assume that the hybrid discrete-continuous robot RL is about to happen.

Shimon Whiteson: Future research in RL will probably place an increasing emphasis on getting humans in the loop. Initial efforts in this direction are already underway, e.g., the TAMER framework, but much remains to be done to exploit developments from the field of human-computer interaction, better understand what kinds of prior knowledge humans can express, and devise methods for learning from the implicit and explicit feedback they can provide. The field of RL could benefit from the development of both richer representations for learning and more practical strategies for exploration. To date, some sophisticated representations such as the indirect encodings used in evolutionary computation can only be used in policy-search methods, as algorithms for using them in value-function methods have not been developed. In addition, even the most efficient strategies for exploration are much too dangerous for many realistic tasks. An important goal is the development of practical strategies for safely exploring in tasks with substantial risk of catastrophic failure. These two areas are intimately connected, as representations should be designed to guide exploration (e.g., by estimating their own uncertainties) and exploration strategies should consider how the samples they acquire will be incorporated into the representation.

Acknowledgements. We want to thank Jürgen Schmidhuber for some help with writing this concluding chapter.

References

- Alpaydin, E.: Introduction to Machine learning. The MIT Press (2010)
 Azar, M.G., Munos, R., Ghavamzadeh, M., Kappen, H.J.: Speedy Q-learning. Advances in Neural Information Processing Systems (2011)

- Bakker, B., Schmidhuber, J.: Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In: Proceedings of the 8th Conference on Intelligent Autonomous Systems, IAS-8, pp. 438–445 (2004)
- Baxter, J., Tridgell, A., Weaver, L.: Knightcap: A chess program that learns by combining $\text{TD}(\lambda)$ with minimax search. Tech. rep., Australian National University, Canberra (1997)
- Berliner, H.: Experiences in evaluation with BKG - a program that plays backgammon. In: Proceedings of IJCAI, pp. 428–433 (1977)
- Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-dynamic Programming. Athena Scientific, Belmont (1996)
- Bishop, C.: Pattern Recognition and Machine learning. Springer, Heidelberg (2006)
- Coates, A., Abbeel, P., Ng, A.: Apprenticeship learning for helicopter control. *Commun. ACM* 52(7), 97–105 (2009)
- Coulom, R.: Efficient Selectivity and Backup Operators in Monte-carlo Tree Search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.(J.) (eds.) CG 2006. LNCS, vol. 4630, pp. 72–83. Springer, Heidelberg (2007)
- Cramer, N.L.: A representation for the adaptive generation of simple sequential programs. In: Grefenstette, J. (ed.) Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 183–187 (1985)
- Crites, R., Barto, A.: Improving elevator performance using reinforcement learning. In: Touretzky, D., Mozer, M., Hasselmo, M. (eds.) Advances in Neural Information Processing Systems, Cambridge, MA, vol. 8, pp. 1017–1023 (1996)
- Di Caro, G., Dorigo, M.: An adaptive multi-agent routing algorithm inspired by ants behavior. In: Proceedings of PART 1998 - Fifth Annual Australasian Conference on Parallel and Real-Time Systems (1998)
- Dietterich, T., Wang, X.: Batch value function approximation via support vectors. In: Advances in Neural Information Processing Systems, vol. 14, pp. 1491–1498 (2002)
- Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. *Evolutionary Computation* 1(1), 53–66 (1997)
- Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B* 26(1), 29–41 (1996)
- Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6, 503–556 (2005)
- Gambardella, L.M., Taillard, E., Dorigo, M.: Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society* 50, 167–176 (1999)
- van Hasselt, H.: Double Q-learning. In: Advances in Neural Information Processing Systems, vol. 23, pp. 2613–2621 (2010)
- Hutter, M.: Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability. Springer, Berlin (2004)
- Kolmogorov, A.: Three approaches to the quantitative definition of information. *Problems of Information Transmission* 1, 1–11 (1965)
- Koza, J.R.: Genetic evolution and co-evolution of computer programs. In: Langton, C., Taylor, C., Farmer, J.D., Rasmussen, S. (eds.) *Artificial Life II*, pp. 313–324. Addison Wesley Publishing Company (1992)
- Koza, J.R.: Genetic Programming II – Automatic Discovery of Reusable Programs. MIT Press (1994)
- Li, M., Vitányi, P.M.B.: An introduction to Kolmogorov complexity and its applications. In: van Leeuwen, J. (ed.) *Handbook of Theoretical Computer Science*, pp. 188–254. Elsevier Science Publishers B.V (1990)

- Littman, M., Boyan, J.: A distributed reinforcement learning scheme for network routing. In: Alspector, J., Goodman, R., Brown, T. (eds.) *Proceedings of the First International Workshop on Applications of Neural Networks to Telecommunication*, Hillsdale, New Jersey, pp. 45–51 (1993)
- Lowe, D.: Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 315–333 (2004)
- Maei, H., Szepesvari, C., Bhatnagar, S., Sutton, R.: Toward off-policy learning control with function approximation. In: *Proceedings of the International Conference on Machine Learning*, pp. 719–726 (2010)
- McCallum, R.A.: Instance-based utile distinctions for reinforcement learning with hidden state. In: Prieditis, A., Russell, S. (eds.) *Machine Learning: Proceedings of the Twelfth International Conference*, pp. 387–395. Morgan Kaufmann Publishers, San Francisco (1995)
- McGovern, A., Andrew, G., Barto, E.M.: Scheduling straight-line code using reinforcement learning and rollouts. In: *Proceedings of Neural Information Processing Systems*. MIT Press (1999)
- Mitchell, T.M.: *Machine learning*. McGraw Hill, New York (1996)
- Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13, 103–130 (1993)
- Moriarty, D.E., Miikkulainen, R.: Efficient reinforcement learning through symbiotic evolution. *Machine Learning* 22, 11–32 (1996)
- Nevmyvaka, Y., Feng, Y., Kearns, M.: Reinforcement learning for optimized trade execution. In: *Proceedings of the 23rd International Conference on Machine Learning*, pp. 673–680 (2006)
- Nouri, A., Littman, M.: Dimension reduction and its application to model-based exploration in continuous spaces. *Machine Learning* 81(1), 85–98 (2010)
- van Otterlo, M.: Efficient reinforcement learning using relational aggregation. *Proceedings of the Sixth European Workshop on Reinforcement Learning, EWRL-6* (2003)
- Peters, J., Mülling, K., Altun, Y.: Relative entropy policy search. In: Fox, M., Poole, D. (eds.) *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010* (2010)
- Poland, J., Hutter, M.: Universal learning of repeated matrix games. In: *Proc. 15th Annual Machine Learning Conf. of Belgium and The Netherlands (Benelearn 2006)*, Ghent, pp. 7–14 (2006)
- Riedmiller, M.: Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) *ECML 2005. LNCS (LNAI)*, vol. 3720, pp. 317–328. Springer, Heidelberg (2005)
- Riedmiller, S., Riedmiller, M.: A neural reinforcement learning approach to learn local dispatching policies in production scheduling. In: *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI 1999)* (1999)
- Ring, M.: Continual learning in reinforcement environments. PhD thesis, University of Texas, Austin, Texas (1994)
- Salustowicz, R.P., Schmidhuber, J.H.: Probabilistic incremental program evolution. *Evolutionary Computation* 5(2), 123–141 (1997)
- Schmidhuber, J.: The Speed Prior: A New Simplicity Measure Yielding Near-Optimal Computable Predictions. In: Kivinen, J., Sloan, R.H. (eds.) *COLT 2002. LNCS (LNAI)*, vol. 2375, pp. 216–228. Springer, Heidelberg (2002)
- Schmidhuber, J.: Optimal ordered problem solver. *Machine Learning* 54, 211–254 (2004)

- Schmidhuber, J.: Ultimate cognition *à la* Gödel. *Cognitive Computation* 1(2), 177–193 (2009)
- Schmidhuber, J.: Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development* 2(3), 230–247 (2010)
- Schmidhuber, J., Zhao, J., Schraudolph, N.: Reinforcement learning with self-modifying policies. In: Thrun, S., Pratt, L. (eds.) *Learning to Learn*, pp. 293–309. Kluwer (1997a)
- Schmidhuber, J., Zhao, J., Schraudolph, N.N.: Reinforcement learning with self-modifying policies. In: Thrun, S., Pratt, L. (eds.) *Learning to Learn*. Kluwer (1997b)
- Schmidhuber, J.H.: Temporal-difference-driven learning in recurrent networks. In: Eckmiller, R., Hartmann, G., Hauske, G. (eds.) *Parallel Processing in Neural Systems and Computers*, pp. 209–212. North-Holland (1990)
- Schmidhuber, J.H.: Curious model-building control systems. In: Proceedings of the International Joint Conference on Neural Networks, vol. 2, pp. 1458–1463. IEEE, Singapore (1991a)
- Schmidhuber, J.H.: A possibility for implementing curiosity and boredom in model-building neural controllers. In: Meyer, J.A., Wilson, S.W. (eds.) *Proceedings of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp. 222–227. MIT Press/Bradford Books (1991b)
- Schmidhuber, J.H.: A general method for incremental self-improvement and multi-agent learning in unrestricted environments. In: Yao, X. (ed.) *Evolutionary Computation: Theory and Applications*. Scientific Publ. Co., Singapore (1996)
- Schmidhuber, J.H., Zhao, J., Wiering, M.A.: Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning* 28, 105–130 (1997c)
- Schoknecht, R.: Optimality of reinforcement learning algorithms with linear function approximation. In: Becker, S., Thrun, S., Obermayer, K. (eds.) *Advances in Neural Information Processing Systems, NIPS 2002*, pp. 1555–1562 (2002)
- van Seijen, H., Whiteson, S., van Hasselt, H., Wiering, M.: Exploiting best-match equations for efficient reinforcement learning. *Journal of Machine Learning Research* 12, 2045–2094 (2011)
- Simpkins, C., Bhat, S., Isbell Jr., C., Mateas, M.: Towards adaptive programming: integrating reinforcement learning into a programming language. *SIGPLAN Not.* 43, 603–614 (2008)
- Singh, S., Litman, D., Kearns, M., Walker, M.: Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *Journal of Artificial Intelligence Research* 16, 105–133 (2002)
- Smart, W., Kaelbling, L.: Effective reinforcement learning for mobile robots. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3404–3410 (2002)
- Solomonoff, R.: A formal theory of inductive inference. Part I. *Information and Control* 7, 1–22 (1964)
- Solomonoff, R.: Complexity-based induction systems. *IEEE Transactions on Information Theory* IT-24(5), 422–432 (1978)
- Sutton, R.S.: Learning to predict by the methods of temporal differences. *Machine Learning* 3, 9–44 (1988)
- Sutton, R.S.: Integrated architectures for learning, planning and reacting based on dynamic programming. In: *Machine Learning: Proceedings of the Seventh International Workshop* (1990)
- Sutton, R.S., Precup, D., Singh, S.P.: Between MDPs and semi-MDPs: Learning, planning, learning and sequential decision making. Tech. Rep. COINS 89-95, University of Massachusetts, Amherst (1998)

- Tanner, B., White, A.: RL-Glue: Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research* 10, 2133–2136 (2009)
- Tesauro, G.: Temporal difference learning and TD-Gammon. *Communications of the ACM* 38, 58–68 (1995)
- Vapnik, V.: *The Nature of Statistical Learning Theory*. Springer, Heidelberg (1995)
- Veness, J., Silver, D., Uther, W., Blair, A.: Bootstrapping from game tree search. In: Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C.K.I., Culotta, A. (eds.) *Advances in Neural Information Processing Systems*, vol. 22, pp. 1937–1945 (2009)
- Veness, J., Ng, K., Hutter, M., Uther, W., Silver, D.: A Monte-carlo AIXI approximation. *Journal of Artificial Intelligence Research* (2011)
- Watkins, C.J.C.H.: Learning from delayed rewards. PhD thesis, King's College, Cambridge, England (1989)
- Watkins, C.J.C.H., Dayan, P.: Q-learning. *Machine Learning* 8, 279–292 (1992)
- Westra, J.: Organizing adaptation using agents in serious games. PhD thesis, Utrecht University (2011)
- Wiering, M.: Self-play and using an expert to learn to play backgammon with temporal difference learning. *Journal of Intelligent Learning Systems and Applications* 2(2), 57–68 (2010)
- Wiering, M., van Hasselt, H.: Ensemble algorithms in reinforcement learning. *IEEE Transactions, SMC Part B, Special Issue on Adaptive Dynamic Programming and Reinforcement Learning in Feedback Control* (2008)
- Wiering, M., van Hasselt, H.: The QV family compared to other reinforcement learning algorithms. In: *Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2009)*, pp. 101–108 (2009)
- Wiering, M.A.: Multi-agent reinforcement learning for traffic light control. In: Langley, P. (ed.) *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 1151–1158 (2000)
- Wiering, M.A., Schmidhuber, J.H.: Solving POMDPs with Levin search and EIRA. In: Saitta, L. (ed.) *Machine Learning: Proceedings of the Thirteenth International Conference*, pp. 534–542. Morgan Kaufmann Publishers, San Francisco (1996)

Index

- E^3 121
- α 30
- α -functions 374
- γ 14
- $\alpha\beta$ -pruning 551
- absorbing state 13, 121
- abstract action
 - sequential 316
- abstract actions 297
- abstraction
 - in Poker 564
- action 10
 - applicable 10
 - precondition 11
- action language 256
- action-selection networks 329
- actor critic 212, 230, 234
 - ADHDP 236
 - Cacla 236, 238
 - natural actor critic 235, 238
- actor only 230
- actor–critic algorithm 32
- Actor-critic methods 369
- agent 5
- agnostic 189
- alternating maximization 481
- approximate linear programming
 - in Tetris 557
- approximate policy iteration 78
 - optimistic 89
- arms race 342
- artificial general intelligence 293
- asymptotic improvement 153
- asynchronous updating 25
- automatic feature construction 563
- average loss 182
- Backgammon 542
 - Neurogammon 543
 - TD-Gammon 543
- backpropagation 543, 546
 - Tangent-Prop 546
- backup operator 21, 24
- Baldwin effect 335
- ball-in-a-cup 599
- basal ganglia 521
- basis function 81
- batch learning 45
 - algorithms 52
 - applications 69
 - problem 46
- Bayesian actor-critic 369
- Bayesian DP 127
- Bayesian game 486
- Bayesian Multi-Task Reinforcement Learning 377
- Bayesian policy gradient 368
- Bayesian Q-learning 361
- Bayesian quadrature 366
- Bayesian reinforcement learning 359
- BEB 128
- BEETLE 127, 376
- behavioral cloning 36
- belief 392
- Belief monitoring 373
- belief state MDP 127
- belief update 392

- Bellman backup 374
 Bellman equation 16, 77, 373
 projected, *see* projected Bellman equation
 Bellman optimality equation 16
 Bellman residual minimization 80, 84
 best response policy 568
 Bgblitz 543
 bias-variance tradeoffs 381
 binary action search 132
 Blocks World 258
 bootstrapping 30
 BOSS 127
 bounded policy iteration 496
 branching factor 542

 Cacla 236, 238
 cart pole
 double-pole cart pole 238
 cart-pole balancing 133
 case-based learning 570
 in real-time strategy games 560
 cellular encodings 343
 Checkers 548, 564
 Chess 545
 knight fork 549
 Morph 564
 NeuroChess 546
 tutoring 566
 UCT 555
 classical conditioning 508
 CMA-ES 221, 238, 330
 in Tetris 557
 cognitive architecture 257
 communication 497
 competing conventions problem 332
 competitions
 AIIDE Starcraft Competition 561
 Ms. Pac-Man Competition 562
 competitive fitness sharing 342
 complexity of dynamic programming 24
 concept drift 9
 confidence interval 552
 counterfactual regret 567
 continual complexification 342
 continuous action space 208
 continuous state space 208
 contraction mapping 24
 CoSyNE 341
 CPPNs 344

 credit assignment 7
 cross-entropy method
 in Tetris 557
 cross-entropy optimization 221, 493
 curse of dimensionality 582

 Darwinian evolution 334
 DBN- E^3 124
 Dec-POMDP 471
 brute force search 480
 decision rule 477
 dynamic programming 489
 factored 494
 history 475
 infinite horizon 495
 model 473
 optimal Q-value function 484
 policy 476
 TOI 494
 transition independent 494
 Dec-POMDP-Com 497
 decentralized POMDP 471
 decision-theoretic planning 18
 decision-theoretic regression 262
 Deep Blue 545
 Deep Fitted Q Iteration 67
 delayed feedback 7
 density estimation 330
 DFQ 67
 difference functions 340
 difficulty scaling 569
 direct policy search 212, 234, 556
 dynamic scripting 559
 partial observability 567
 Dirichlet distribution 372
 Dirichlet mixture 379
 Dirichlet process 380
 discount factor 14
 diversity of gameplay 569
 domain knowledge 360, 553
 in games 561
 dopamine 518
 DRE 132
 DYNA 38
 Dyna 118
 Dyna-2 551
 Dyna-2 119
 Dynamic Bayesian Network 122
 Dynamic Programming 373

- dynamic programming 19
 - Dec-POMDP 489
 - memory-bounded 492
 - point-based 491
- dynamic scripting 570
 - difficulty scaling 569
 - in real-time strategy games 559
- EDI-CR 494
- eligibility trace 37
- eligibility traces 227
 - performance in Backgammon 545
- Elo rating 545
 - of patterns 551
- envelope 26
- environment 5
- episodic task 12
- ESP 341
- estimation of distribution algorithms 330
- evaluative feedback 8
- event-driven Dec-MDP 494
- evolutionary algorithms 221
- evolutionary computation 325
 - coevolution 339
 - neuroevolution 327
 - on-line 346
 - steady-state 338
- evolutionary function approximation 335
- evolutionary game theory 550
- evolutionary learning
 - in Starcraft 560
 - in Tetris 557
 - in Wargus 560
- evolutionary strategies 221, 234
 - CMA-ES 221, 238
 - natural evolutionary strategies 221, 234
- EvolutionChamber 560
- experience replay 50
- experiment
 - double-pole cart pole 238
- explicit fitness sharing 334
- exploration 28, 34, 88, 126
 - automatic 544
 - Boltzmann 29
 - Boltzmann exploration 232
 - ϵ -greedy 565
 - Gaussian exploration 232
 - in games 564
 - softmax 29
- exploration gain 377
- exploration overhead 49
- exploration-exploitation tradeoff 8
- exploration/exploitation tradeoff 360, 373
- factored Dec-POMDP 494
- factored state 122
- factored-r-max 124
- family of relational MDPs 260
- feature transfer 157, 161, 162
- finite horizon task 12
- Finite Sample Analysis 380
- first-order (logical) basis functions 266
- first-order decision-theoretic regression 264
- Fisher kernels 368
- fitness function 325
- Fitted Q Iteration 55
- Fitted-R-Max 132
- fitting 50
- fog of war 566
- forward-sweep policy computation 485
- FQI 55
- fun gameplay 568
 - balanced strategies 571
- function approximation 34, 208, 212, 592
 - discretization 214
 - fuzzy sets 217
 - in Tetris 556
 - linear 213
 - non-linear 217
 - tile coding 214
- game balance 571
- games 539
 - Atari 2600 games 562
 - Backgammon 542
 - Baldur's Gate 569
 - Black & White 570
 - Bridge 567
 - Checkers 564
 - Chess 545
 - Civilization 570
 - Creatures 570
 - Diplomacy 564
 - general gameplay 564
 - Go 550
 - Hearts 564
 - Knock'em 569
 - Last Night on Earth 571
 - Magic: The Gathering 571

- Ms. Pac-Man 564
- NERO 570
- Neverwinter Nights 570, 571
- Othello 564
- Poker 564, 568
- Pong 569
- real-time strategy 558
- Suicide chess 550
- SZ-Tetris 556
- teaching games 570
- Tetris 555
- Gaussian process 363
- Gaussian Process Temporal Difference 363
- generalization 114
- generalized multi-agent A* 488
- generalized policy iteration 18
- generative and developmental systems 327, 343
- generative model 113
- genetic algorithms 325, 493
- GMAA* 488
- GNUGo 543
- GNUChess 547
- Go 550
 - nakade 555
 - tutoring 566
- goal specification 585
- goal state 12
- gradient descent 219
- gradient free optimization 220
- gradient temporal-difference learning 226
- growing batch learning problem 48
- guidance 36
- heuristics 26
 - equivalent experience 553
 - RAVE 553
 - virtual wins/losses 553
- HEXQ
 - exit 315
- hierarchical reinforcement learning (HRL) 295
 - HAMQ 307
 - HEXQ 315
 - MAXQ 309
 - options 306
- host/parasite model 342
- hyperparameters 373
- imitation 594
- incremental policy generation 491
- indefinite horizon task 12
- independent Q-learners 496
- indirect encodings 343
- inductive logic programming 256
- infinite horizon task 12
- initial state distribution 12
- innovation numbers 333
- instance transfer 150
- instructive feedback 8
- intensional dynamic programming 262
- international planning competition 267
- inverse reinforcement learning 585
- iterated elimination of dominated policies 491
- JESP 481
- joint action 445
- joint belief 482, 498
- jumpstart improvement 153
- KADP 52
- kernel-based approximate dynamic programming 52
- KnightCap 547
- known state-action MDP 184, 187, 192, 198
 - empirical known state-action MDP 184
- KWIK 121, 189
 - Knows What It Knows 189
 - KWIK-learnable 190
 - KWIK-learnable MDPs 192
- λ -policy iteration
 - in Tetris 557
- L-systems 343
- Lamarckian evolution 334
- Law of Effect 514
- learning classifier systems 335, 336
 - anticipatory 346
 - Michigan-style 337
 - Pittsburgh-style 337
- learning rate 30
- learning speed improvement 151
- least-squares policy evaluation 84
- least-squares policy iteration 57, 86
 - in Tetris 557
 - online 90
- least-squares temporal difference 84

- lifted first-order reasoning 266
- logical generalization 257
- logical language 258
- logical matching 263
- logical regression 263
- lookahead search
 - multi-step 547
- LSE-R-Max 124
- LSPI 57, 132
- MAA* 488
- Markov 11
- Markov decision process 10, 12
- Markov game
 - definition 455
- Markov property 456
- maximum likelihood estimation 368
- MBBE 128
- MBIE 128
- memory-bounded dynamic programming 492
- Met-R-Max 124
- micromanagement 570
- minimax 342
- minimax search 547, 555
- minimax-optimal strategy 568
- missing information
 - in games 566
- model 12
 - learning of 33, 38
- model approximation 211
- model errors 584
- model-based algorithm 17, 19
- model-based control 516
- Model-Based Reinforcement Learning 372
- model-free algorithm 17, 27
- model-free control 516
- Model-free RL 361
- modified policy iteration 25
- monomial basis 376
- Monte Carlo sampling 33
- Monte Carlo Tree Search 115
- Monte-Carlo 366
 - hidden information 567
 - value estimation 552
- Monte-Carlo tree search 552
 - in Chess 549, 555
 - in Go 553
- Morph 549, 564
- multi-agent A* 488
- multi-agent belief 490, 491
- multi-agent MDP 494
- multi-agent reinforcement learning 496
- Multi-task learning 377
- myopic policy 377
- Nash equilibrium 481, 494
- Nash-equilibrium 568
- natural actor critic 235, 238
- natural evolutionary strategies 221, 234
- natural gradient 220
- natural policy gradient 233
- ND-POMDP 495
- NEAT 333
- Neural Fitted Q Iteration 61
- neural network 543, 546
 - as automatic feature construction 563
 - as predictive model 546
 - for pattern evaluation 551
- neural networks 327
 - feed-forward 329
 - recurrent 329
- NeuroChess 546
- Neurogammon 543
- neurons 340
- Neverwinter Nights 570, 571
- NFQ 61
- niching 332
- object 253, 257
- object-oriented RL 137
- objective functions 224
- observation model 390
- off-policy learning 32
- offline learning 7
- on-policy learning 31
- Online algorithms 376
- online learning 7
- operant conditioning 513
- opponent modelling 567
- optimal
 - hierarchical greedy 304
 - hierarchically optimal 304
 - recursively optimal 304
- optimism in the face of uncertainty 181, 186, 197
- optimistic value initialization 29
- option transfer 156, 161

- ORTS 559
- ORTS Competition 561
- Othello 564
- PAC-Bayesian approach 382
- PAC-MDP 179, 382
- parallel tasks 558
- parameter transfer 151
- parameterized action 257
- Pareto optimal 494
- Pareto optimality 342
- parti-game 131
- partial observability 388
- partial visibility 566
- partially observable Markov decision process 11, 372, 390
- partially observable stochastic game 493
- patterns 551, 554
 - local 549, 551
 - move patterns 549
 - team of 551
- perceptual aliasing 388
- Perseus 376
- PIAGeT 254
- PILCO 131
- planning 6, 18, 115
- player ranking
 - Elo rating 545
 - rollout analysis 543
 - tournament 543
- playtesting 571
- point-based dynamic programming
 - Dec-POMDP 491
- point-based value iteration 376
- Poker 564, 568
 - state aggregation 564
- policy 13
 - ε -greedy 29
 - application of 13
 - deterministic 13
 - greedy 17, 21
 - optimal 13, 16
 - stochastic 13
- policy approximation 212, 229
- policy evaluation 18, 20, 77
 - least-squares, *see* least-squares policy evaluation
 - projected, *see* projected policy evaluation
- Policy gradient 365
- policy gradient 230, 366
 - natural policy gradient 233
- policy gradient theorem 370
- policy improvement 18, 21, 78
- policy iteration 20, 77
 - approximate, *see* approximate policy iteration
 - least-squares, *see* least-squares policy iteration
- policy search 588
- POMDP 390
 - Bellman equation 394
 - learning internal memory 405
 - model-based techniques 395
 - model-free techniques 404
 - point-based methods 401
 - value function 394
 - value iteration 398
- POMDPs
 - deep-memory 341
- Pong 569
- POSG 493
- posterior Gaussian process 364
- preference function 556, 557
- Prior Knowledge 379
- prior knowledge 594
- prioritized sweeping 33, 38
- probabilistic inference for planning 280
- probabilistic model-building genetic algorithms 330
- probabilistic programming languages 281
- probabilistic STRIPS operator 260
- probimax search 568
- progressive unpruning/widening 553
- projected Bellman equation 80
- projected policy evaluation 80
- projections 224
 - projected Bellman equation 225
- Q-function 17
- Q-learning 31, 227
 - difficulty scaling 569
 - in Chess 546
- QBG 487
- QMDP 487
- QPOMDP 487
- quiescence search 547
- R-IAC 130
- R-Max 121
- randomness 544, 556

- RAVE 553
real-time architecture 119
real-time dynamic programming 27
real-time strategy game 558
 difficulty scaling 569
 knight's rush 559
 seven-roach rush 560
real-world interactions 584
real-world samples 583
realizable 189
regret 180
reinforcement learning 27
 in commercial games 570
relation 253, 257
relational
 decision list 261
 decision tree 260
 interpretation 259
 learning world models 277
 Markov decision process 259
 policy 261
 policy search 274
 POMDP 282
 reward function 260
 value function 260
relational reinforcement learning 254, 268
relational representation 257
 in games 563
representation 9, 589
 automatic feature construction 563
 combination features 563
 expert features 563
 hard to learn 564
 in Backgammon 544
 in Chess 546
 relational 563
representation transfer 150
Rescorla-Wagner model 511
reward
 local reward 551
reward function 11, 12
reward model
 average reward 14
 finite horizon 14
 infinite horizon 14
rich evaluative feedback 562
risk-sensitive optimization 381
RL-DT 125
robot 579
robust control 381
roll-out 115
rollout 552
rollout analysis 543
rollout policy 553
 balanced 554
RSPSA 565
sample complexity of exploration 178
sample efficiency 120
SANE 340
SARSA 31
Sarsa 227
 in Neverwinter Nights 571
search 26
self-play 543, 547, 548, 565
semantics of communication 497
semi Markov Decision Problem (SMDP)
 297
sequence form 491, 493
sequential decision making 5
sGA 331
shaping 36
simulation 596
simulation balancing 554
SLF-R-Max 124
Snowie 543
sparse sampling 116, 377
speciation 332
SPITI 125
Starcraft 558
state 10
 value 15
state abstraction 301
 eliminating irrelevant variables 301
 funnelling 302
state aggregation 563
state representation
 atomic 255
 deictic 255
 propositional 255
 relational 255
state-action value function 17
statistical relational learning 256
stochastic bisimulation homogeneity 302
Stratagus 558
structural credit assignment 8
structure learning 123
structured Bellman backup 262
sub-tree policy 478

- substitution 259
- subsumption 259
- Suicide chess 550
- synchronous updating 25
- SZ-Tetris 556, 558
- tabular model 113
- targeted exploration 112
- task-hierarchy 300
 - partial-order 316
- taxi 123
- TD-Gammon 543
 - vs. humans 543
- TD-Leaf 547
 - exploration 565
- TD-learning
 - in Magic: The Gathering 571
- TD-POMDP 495
- temporal credit assignment 7
- temporal credit assignment problem 27
- temporal difference 511
- temporal difference learning 29, 543
 - and tree search 547
 - of value function slope 546
 - $\text{TD}(\lambda)$ 543
- temporal-difference learning 226
 - GTD2 228
 - Q-learning 227
 - Sarsa 227
 - TD-learning 226
 - TDC 228
- terminal state 13
- Tetris 555
 - NP-hardness 556
- TEXPLORE 129
- Thompson sampling 377
- tile coding 214
- time-to-go 478
- TOI-Dec-MDP 494
- topology 331
- training regime 565
 - database play 547, 566
 - expert training 548
- lesson and practice 566
- online game server 548
- self-play 543, 547, 548, 565
- tutoring 566
- transfer learning 317
- transition function 11, 12
 - Markovian 11
- tree search
 - $\alpha\beta$ -pruning 551
 - principal leaf 547
 - TD-Leaf 547
- TreeStrap 548
- tutoring 566
- TWEANN 331
- UCT 117, 552, *see also* Monte-Carlo tree search
 - exploration 565
 - narrow path to victory 554
 - shallow traps 555
- underlying MDP 487
- underlying POMDP 487
- Utile Coordination 462
- value approximation 211, 223
 - off-policy 223
 - offline 223
 - on-policy 223
 - online 223
- value function
 - as probability of victory 545
- value function 15
 - vs. preference function 556, 557
- value function approximation
 - weak performance in Tetris 556
- value function decomposition 303
- value iteration 23
- value of perfect information 128, 362, 377
- Vexbot 568
- Warcraft 558
- XCS 337