Rollout, Policy Iteration, and Distributed Reinforcement Learning

Dimitri P. Bertsekas

Athena Scientific

Rollout, Policy Iteration, and Distributed Reinforcement Learning

by

Dimitri P. Bertsekas

Arizona State University
and
Massachusetts Institute of Technology

WWW site for book information and orders

http://www.athenasc.com

Athena Scientific, Belmont, Massachusetts

Athena Scientific
Post Office Box 805
Nashua, NH 03060
U.S.A.

Email: info@athenasc.com
WWW: http://www.athenasc.com

Cover photography by Dimitri Bertsekas.
Stars over the Stata Center at MIT (built on the location of the old Building 20 where Claude Shannon had his first office as a professor in 1956).

Publisher's Cataloging-in-Publication Data

Bertsekas, Dimitri P.
Rollout, Policy Iteration, and Distributed Reinforcement Learning
Includes Bibliography and Index
1. Mathematical Optimization. 2. Dynamic Programming. I. Title.
QA402.5 .B465 2020 519.703 00-91281

ISBN-10: 1-886529-07-8, ISBN-13: 978-1-886529-07-6

2nd Printing (includes revisions and updates)

ABOUT THE AUTHOR

Dimitri Bertsekas studied Mechanical and Electrical Engineering at the National Technical University of Athens, Greece, and obtained his Ph.D. in system science from the Massachusetts Institute of Technology. He has held faculty positions with the Engineering-Economic Systems Department, Stanford University, and the Electrical Engineering Department of the University of Illinois, Urbana. Since 1979 he has been teaching at the Electrical Engineering and Computer Science Department of the Massachusetts Institute of Technology (M.I.T.), where he is McAfee Professor of Engineering. In 2019, he joined the School of Computing, Informatics, and Decision Systems Engineering at the Arizona State University, Tempe, AZ, as Fulton Professor of Computational Decision Making.

Professor Bertsekas' teaching and research have spanned several fields, including deterministic optimization, dynamic programming and stochastic control, large-scale and distributed computation, artificial intelligence, and data communication networks. He has authored or coauthored numerous research papers and eighteen books, several of which are currently used as textbooks in MIT classes, including "Dynamic Programming and Optimal Control," "Data Networks," "Introduction to Probability," and "Nonlinear Programming."

Professor Bertsekas was awarded the INFORMS 1997 Prize for Research Excellence in the Interface Between Operations Research and Computer Science for his book "Neuro-Dynamic Programming" (co-authored with John Tsitsiklis), the 2001 AACC John R. Ragazzini Education Award, the 2009 INFORMS Expository Writing Award, the 2014 AACC Richard Bellman Heritage Award, the 2014 INFORMS Khachiyan Prize for Life-Time Accomplishments in Optimization, the 2015 MOS/SIAM George B. Dantzig Prize, and the 2022 IEEE Control Systems Award. In 2018 he shared with his coauthor, John Tsitsiklis, the 2018 INFORMS John von Neumann Theory Prize for the contributions of the research monographs "Parallel and Distributed Computation" and "Neuro-Dynamic Programming." Professor Bertsekas was elected in 2001 to the United States National Academy of Engineering for "pioneering contributions to fundamental research, practice and education of optimization/control theory, and especially its application to data communication networks."

ATHENA SCIENTIFIC

OPTIMIZATION AND COMPUTATION SERIES

1. Rollout, Policy Iteration, and Distributed Reinforcement Learning, by Dimitri P. Bertsekas, 2020, ISBN 978-1-886529-07-6, 480 pages
2. Reinforcement Learning and Optimal Control, by Dimitri P. Bertsekas, 2019, ISBN 978-1-886529-39-7, 388 pages
3. Abstract Dynamic Programming, 2nd Edition, by Dimitri P. Bertsekas, 2018, ISBN 978-1-886529-46-5, 360 pages
4. Dynamic Programming and Optimal Control, Two-Volume Set, by Dimitri P. Bertsekas, 2017, ISBN 1-886529-08-6, 1270 pages
5. Nonlinear Programming, 3rd Edition, by Dimitri P. Bertsekas, 2016, ISBN 1-886529-05-1, 880 pages
6. Convex Optimization Algorithms, by Dimitri P. Bertsekas, 2015, ISBN 978-1-886529-28-1, 576 pages
7. Convex Optimization Theory, by Dimitri P. Bertsekas, 2009, ISBN 978-1-886529-31-1, 256 pages
8. Introduction to Probability, 2nd Edition, by Dimitri P. Bertsekas and John N. Tsitsiklis, 2008, ISBN 978-1-886529-23-6, 544 pages
9. Convex Analysis and Optimization, by Dimitri P. Bertsekas, Angelia Nedić, and Asuman E. Ozdaglar, 2003, ISBN 1-886529-45-0, 560 pages
10. Network Optimization: Continuous and Discrete Models, by Dimitri P. Bertsekas, 1998, ISBN 1-886529-02-7, 608 pages
11. Network Flows and Monotropic Optimization, by R. Tyrrell Rockafellar, 1998, ISBN 1-886529-06-X, 634 pages
12. Introduction to Linear Optimization, by Dimitris Bertsimas and John N. Tsitsiklis, 1997, ISBN 1-886529-19-1, 608 pages
13. Parallel and Distributed Computation: Numerical Methods, by Dimitri P. Bertsekas and John N. Tsitsiklis, 1997, ISBN 1-886529-01-9, 718 pages
14. Neuro-Dynamic Programming, by Dimitri P. Bertsekas and John N. Tsitsiklis, 1996, ISBN 1-886529-10-8, 512 pages
15. Constrained Optimization and Lagrange Multiplier Methods, by Dimitri P. Bertsekas, 1996, ISBN 1-886529-04-3, 410 pages
16. Stochastic Optimal Control: The Discrete-Time Case, by Dimitri P. Bertsekas and Steven E. Shreve, 1996, ISBN 1-886529-03-5, 330 pages

Contents

Preface

We know the past but cannot control it. We control the future but cannot know it.

Claude Shannon

In this research monograph we discuss the solution of large and challenging multistage decision problems using methods of reinforcement learning (RL for short), also referred to by other names such as approximate dynamic programming and neuro-dynamic programming. We will focus on a subset of methods which are based on the idea of *policy iteration*, i.e., starting from some policy and generating one or more improved policies.

If just one improved policy is generated, this is called *rollout*, which, based on broad and consistent computational experience, appears to be one of the simplest and most reliable of all RL methods. Rollout is also well-suited for on-line model-free implementation and on-line replanning. Approximate policy iteration can be viewed as repeated application of rollout. This is one of the most prominent types of RL methods. It can be implemented using data generated by the system itself, a process known as *self-learning*, and value and policy approximation architectures, including neural networks. Both rollout and policy iteration are related to the classical Newton's method for iterative optimization, which in turn explains their associated large cost improvements and fast convergence.

Approximate policy iteration is more ambitious than rollout, but it is a strictly off-line method, and it is generally far more computationally intensive (of course rollout may also require a lot of on-line computation). This motivates the use of parallel and distributed computation. One of the purposes of the monograph is to discuss distributed (possibly asynchronous) methods that relate to rollout and policy iteration, both in the context of an exact and an approximate implementation involving neural networks or other approximation architectures.

One of the contributions of the monograph is to develop variants of rollout and policy iteration for problems with a multiagent structure, where the control consists of multiple components, each associated with a separate agent. In particular, we introduce a new approach to lookahead simplification through the use of *multiagent rollout*, which allows the dra-

matic reduction of the computational requirements for one-step lookahead when the control consists of multiple components, and connects with the theme of distributed asynchronous implementation.

Multiagent rollout also has a strong connection with a well-developed body of research with a long history: the theory of teams and the notion of person-by-person optimality. In particular, we develop an infinite horizon dynamic programming methodology, which includes value and policy iteration methods that converge to a person-by-person optimal policy. While our multiagent schemes are based on fully shared agent information, they are also well suited as a starting point for approximations, in the context of on-line autonomous decision making by multiple agents each coordinating in varying degrees with the other agents. In this context, agent information that is not shared by other agents, is appropriately estimated, with the estimates being treated as if they were exact.

Several of the ideas that we develop in some depth in this monograph have been central in the implementation of recent high profile successes, such as the AlphaZero program for playing chess, Go, and other games. In addition to the fundamental process of successive policy iteration/improvement, this program includes the use of deep neural networks for representation of both value functions and policies, the extensive use of large scale parallelization, and the simplification of lookahead minimization, through methods involving Monte Carlo tree search and pruning of the lookahead tree. In this monograph, we also focus on policy iteration, value and policy neural network representations, parallel and distributed computation, and lookahead simplification. Thus while there are significant differences, the principal design ideas that form the core of this monograph are shared by the AlphaZero architecture, except that we develop these ideas in a broader and less application-specific framework.

Another subject that we deal with in some detail is model predictive control (MPC for short), one of the most prominent control system design methods at present. One of the reasons is that classical forms of MPC are closely related to (and indeed can be viewed as) rollout algorithms, thereby providing a connection with reinforcement learning, which is beneficial in two ways. On one hand the MPC context provides rich crossfertilization opportunities with the analytical and algorithmic ideas of rollout and RL; for example the notion of sequential improvement in rollout is intimately connected to Lyapunov stability analysis in MPC, and the target tube ideas that are central in MPC may prove useful in the context of constrained rollout and policy iteration. On the other hand the dynamic programming and RL methodologies point the way to extensions of MPC based on self-learning, approximate policy iteration, simulation, the treatment of stochastic and set membership uncertainty, and the use of distributed computation.

In our development of several of the topics of this book we rely on methodology that is covered in greater depth in the 1996 neuro-dynamic

programming book [BeT96] (jointly written with J. Tsitsiklis) as well as the author's recent RL book [Ber19a]. However, we aim to develop rollout and approximate policy iteration beyond the books [BeT96] and [Ber19a]. In particular, we present new research, relating to rollout variants, distributed asynchronous computation, partitioned architectures, and multiagent systems. We also indicate how our methods are well-suited for several types of challenging large scale optimization problems, such as combinatorial/discrete optimization, as well as partially observed Markov decision problems (POMDP).

This monograph took shape in the fall of 2019 and was largely based on two separate but related lines of research for distributed large-scale computation:

(a) My work on multiagent rollout, policy iteration, and value iteration, which was published in the papers [Ber19c], [Ber19d], [Ber20], [Ber21a]. It was based on my earlier work on rollout, which started in the mid 90s, in the context of the neuro-dynamic programming book, and continued for several years afterwards.

(b) My work on distributed policy iteration algorithms with state space-partitioned architectures. These ideas were extended, implemented, and applied to some large-scale POMDP problems in collaboration with my Arizona State University (ASU) colleagues Sushmita Bhattacharya, Sahil Badyal, Thomas Wheeler, and Stephanie Gil [BBW20]. This work is also connected with my joint research on asynchronous distributed state space-partitioned policy iteration with Huizhen Yu [BeY10], [BeY12], [YuB13], which is presented in Section 5.8 of this monograph.

Most of the book was written while teaching a research-oriented course at ASU, starting in January 2020. The hospitable and stimulating environment at ASU contributed much to my productivity during this period, and for this I am very thankful to several colleagues and students, including Stephanie Gil, Giulia Pedrielli, and Petr Sulc, and my teaching assistant, Sushmita Bhattacharya. I have also appreciated fruitful interactions with colleagues and students outside ASU, particularly Yuchao Li, who also provided valuable proofreading support.

Finally, I would like to dedicate this monograph to the creative genius of Claude Shannon, the father of information theory, but also the father of computer chess. His approximate dynamic programming ideas, which predated the work of Bellman, live on inside the AlphaZero program, the most impressive success story of reinforcement learning up to now.

Dimitri P. Bertsekas
July 2020

NOTE ABOUT THIS UPDATED PRINTING

This 2nd printing was prompted by the publication of the book in China, and by the use of the book for an on-line course at ASU in the Spring of 2021. See my website:

http://web.mit.edu/dimitrib/www/RLbook.html

Simultaneously with its publication in China, this 2nd printing will replace the 1st printing outside of China.

In addition to editorial corrections, I took the opportunity to make some more substantive revisions. One type of revision aimed to enrich the material on multiagent control systems, and to introduce enhancements in the form of signaling policies (see Section 5.3.4). Another type of revision aimed to highlight the connections of AlphaZero and related programs with approximations in value space and the on-line play idea that lies at the heart of rollout.

The significance of on-line policy improvement by multistep lookahead and rollout came into focus with the success of AlphaZero and the earlier, but just as impressive TD-Gammon program. Both of these programs involve an *off-line training* algorithm, and an *on-line play* algorithm that relies on the results of the off-line training. These two algorithms are different, but a key fact is that *the on-line player performs much better than the extensively trained off-line player*; see Section 1.1.

In practical problems of decision and control, a lot of analysis and/or off-line computation is often directed towards obtaining a policy, which is inevitably suboptimal, because of model imperfections, changing problem parameters, and overwhelming computational bottlenecks. The AlphaZero and TD-Gammon experience reinforces an important conclusion: despite the off-line effort that may have gone into the design of a policy, its performance may be greatly improved by on-line approximation in value space, with long lookahead (involving minimization or rollout with this policy), and terminal cost approximation.

This performance enhancement by on-line play goes well beyond the conventional control wisdom that "feedback corrects for noise, uncertainty, and modeling errors." It is embodied to some extent by the model predictive control methodology, but it also suggests a more broadly applicable paradigm, whose significance has yet to be fully recognized by the decision and control community.

In this revised printing I have also aimed to illustrate the ideas of on-line play, rollout, and policy iteration with intuitive figures that draw upon the use of abstract forms of the Bellman equation operators. In this way a direct line, couched on insightful visualization, can be drawn from AlphaZero to optimal, model predictive, and adaptive control.

To improve the didactic value of the book, I have also added some material on infinite horizon problems to Chapter 1. Furthermore, I have added exercises for the reader at the end of each chapter. Moreover, during the preceding year I supplemented the book with quite a few on-line extensions, including research papers, and lecture slides and videos. In particular, I posted a series of videolectures and slides from my 2021 course on reinforcement learning at ASU. This material is freely accessible from my website. The videolectures are also available at

https://www.youtube.com/playlist?list
=PLmH30BG15SIp79JRJ-MVF12uvB1qPtPzn

and at

https://space.bilibili.com/2036999141

Finally, I wish to thank students and colleagues for many helpful comments relating to the 1st printing and the ASU course. I am particularly thankful to Yuchao Li for proofreading support and numerous valuable suggestions.

Dimitri P. Bertsekas

July 2021

1

Exact and Approximate Dynamic Programming Principles

Contents

In this chapter, we provide some background on exact dynamic programming (DP), with a view towards the suboptimal solution methods, which are based on approximation in value space and are the main subject of this book. We first discuss finite horizon problems, which involve a finite sequence of successive decisions, and are thus conceptually and analytically simpler. We then consider somewhat briefly the more intricate infinite horizon problems, but defer a more detailed treatment for Chapter 5.

We will discuss separately deterministic and stochastic problems (Sections 1.2 and 1.3, respectively). The reason is that deterministic problems are simpler and have some favorable characteristics, which allow the application of a broader variety of methods. Significantly they include challenging discrete and combinatorial optimization problems, which can be fruitfully addressed with some of the rollout and approximate policy iteration methods that are the main focus of this book.

In subsequent chapters, we will discuss selectively some major algorithmic topics in approximate DP and reinforcement learning (RL), including rollout and policy iteration, multiagent problems, and distributed algorithms. A broader discussion of DP/RL may be found in the author's RL textbook [Ber19a], and the DP textbooks [Ber12], [Ber17a], [Ber18a], the neuro-dynamic programming monograph [BeT96], as well as the literature cited in the last section of this chapter.

The DP/RL methods that are the principal subjects of this book, rollout and policy iteration, have a strong connection with the famous AlphaZero, AlphaGo, and other related programs. As an introduction to our technical development, we take a look at this connection in the next section.

1.1 ALPHAZERO, OFF-LINE TRAINING, AND ON-LINE PLAY

One of the most exciting recent success stories in RL is the development of the AlphaGo and AlphaZero programs by DeepMind Inc; see [SHM16], [SHS17], [SSS17]. AlphaZero plays Chess, Go, and other games, and is an improvement in terms of performance and generality over AlphaGo, which plays the game of Go only. Both programs play better than all competitor computer programs available in 2020, and much better than all humans. These programs are remarkable in several other ways. In particular, they have learned how to play without human instruction, just data generated by playing against themselves. Moreover, they learned how to play very quickly. In fact, AlphaZero learned how to play chess better than all humans and computer programs within hours (with the help of awesome parallel computation power, it must be said).

Perhaps the most impressive aspect of AlphaZero/chess is that its play is not just better, but it is also very different than human play in terms of long term strategic vision. Remarkably, AlphaZero has discovered

new ways to play chess, a game that has been studied intensively by humans for hundreds of years.

Still, for all of its impressive success and brilliant implementation, AlphaZero is couched on well established methodology, which is the subject of the present book, and is portable to far broader realms of engineering, economics, and other fields. This is the methodology of DP, policy iteration, limited lookahead, rollout, and approximation in value space.†

To understand the overall structure of AlphaZero and related programs, and their connections to our DP/RL methodology, it is useful to divide their design into two parts:

(a) *Off-line training*, which is an algorithm that learns how to evaluate chess positions, and how to steer itself towards good positions with a default/base chess player.

(b) *On-line play*, which is an algorithm that generates good moves in real time against a human or computer opponent, using the training it went through off-line.

We will next briefly describe these algorithms, and relate them to DP concepts and principles.

Off-Line Training and Policy Iteration

An off-line training algorithm like the one used in AlphaZero is the part of the program that learns how to play through self-training that takes place before real-time play against any opponent. It is illustrated in Fig. 1.1.1, and it generates a sequence of *chess players* and *position evaluators*. A chess player assigns "probabilities" to all possible moves at any given chess position (these are the probabilities with which the player selects the possible moves at the given position). A position evaluator assigns a numerical score to any given chess position (akin to a "probability" of winning the game from that position), and thus predicts quantitatively the performance of a player starting from any position. The chess player and the position evaluator are represented by two neural networks, a *policy*

† It is also worth noting that the principles of the AlphaZero design have much in common with the work of Tesauro [Tes94], [Tes95], [TeG96] on computer backgammon. Tesauro's programs stimulated much interest in RL in the middle 1990s, and exhibit similarly different and better play than human backgammon players. A related impressive program for the (one-player) game of Tetris, also based on the method of policy iteration, is described by Scherrer et al. [SGG15], together with several antecedents. Also the AlphaZero ideas have been replicated by the publicly available program Leela Chess Zero, with similar success. For a better understanding of the connections of AlphaZero, Tesauro's programs (TD-Gammon [Tes94], and its rollout version [TeG96]), and the concepts developed here, the reader may consult the "Methods" section of the paper [SSS17].

network and a *value network*, which accept a chess position and generate a set of move probabilities and a position evaluation, respectively.†

In the more conventional DP-oriented terms of this book, a position is the state of the game, a position evaluator is a cost function that gives the cost-to-go at a given state, and the chess player is a randomized policy for selecting actions/controls at a given state.‡

The overall training algorithm is a form of *policy iteration*, a DP algorithm that will be of primary interest to us in this book. Starting from a given player, it repeatedly generates (approximately) improved players, and settles on a final player that is judged empirically to be "best" out of all the players generated.†† Policy iteration may be separated conceptually into two stages (see Fig. 1.1.1).

(a) *Policy evaluation*: Given the current player and a chess position, the outcome of a game played out from the position provides a single data point. Many data points are thus collected, and are used to train a value network, whose output serves as the position evaluator for that player.

---

† Here the neural networks play the role of *function approximators*. By viewing a player as a function that assigns move probabilities to a position, and a position evaluator as a function that assigns a numerical score to a position, the policy and value networks provide approximations to these functions based on training with data (training algorithms for neural networks and other approximation architectures will be discussed in Chapter 4). Actually, AlphaZero uses the same neural network for training both value and policy. Thus there are two outputs of the neural net: value and policy. This is pretty much equivalent to having two separate neural nets and for the purpose of the book, we prefer to explain the structure as two separate networks. AlphaGo uses two separate value and policy networks. Tesauro's backgammon programs use a single value network, and generate moves when needed by one-step or two-step lookahead minimization, using the value network as terminal position evaluator.

‡ One more complication is that chess and Go are two-player games, while most of our development will involve single-player optimization. However, DP theory and algorithms extend to two-player games, although we will not discuss these extensions, except briefly in Chapters 3 and 5 (see Sections 3.6 and 5.5).

†† Quoting from the paper [SSS17]: "The AlphaGo Zero selfplay algorithm can similarly be understood as an approximate policy iteration scheme in which MCTS is used for both policy improvement and policy evaluation. Policy improvement starts with a neural network policy, executes an MCTS based on that policy's recommendations, and then projects the (much stronger) search policy back into the function space of the neural network. Policy evaluation is applied to the (much stronger) search policy: the outcomes of selfplay games are also projected back into the function space of the neural network. These projection steps are achieved by training the neural network parameters to match the search probabilities and selfplay game outcome respectively."

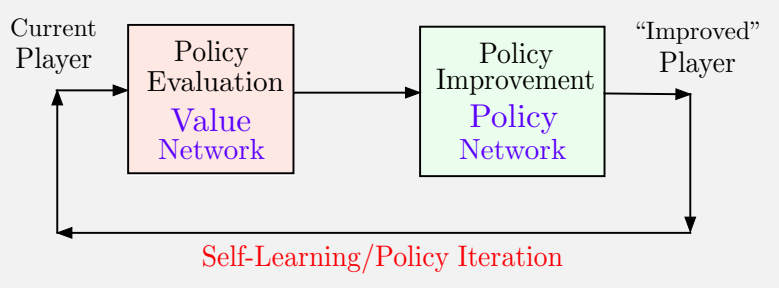


Figure 1.1.1 Illustration of the AlphaZero off-line training algorithm. It generates a sequence of position evaluators and chess players. The position evaluator and the chess player are represented by two neural networks, a value network and a policy network, which accept a chess position and generate a position evaluation and a set of move probabilities, respectively.

(b) *Policy improvement*: Given the current player and its position evaluator, trial move sequences are selected and evaluated for the remainder of the game starting from many positions. An improved player is then generated by adjusting the move probabilities of the current player towards the trial moves that have yielded the best results. In AlphaZero this is done with a complicated algorithm called *Monte Carlo Tree Search*, which will be described in Chapter 2. However, policy improvement can be done more simply. For example one could try all possible move sequences from a given position, extending forward to a given number of moves, and then evaluate the terminal position with the player's position evaluator. The move evaluations obtained in this way are used to nudge the move probabilities of the current player towards more successful moves, thereby obtaining data that is used to train a policy network that represents the new player.

On-Line Play and Approximation in Value Space - Rollout

Consider now the "final" player obtained through the AlphaZero off-line training process. It can play against any opponent by generating move probabilities at any position using its off-line trained policy network, and then simply play the move of highest probability. This player would play very fast on-line, but it would not play good enough chess to beat strong human opponents. The extraordinary strength of AlphaZero is attained only after the player obtained from off-line training is embedded into another algorithm, which we refer to as the "on-line player."† In other words *AlphaZero plays on-line much better than the best player it has produced*

† Quoting from the paper [SSS17]: "The MCTS search outputs probabilities of playing each move. These search probabilities usually select much stronger moves than the raw move probabilities of the neural network."

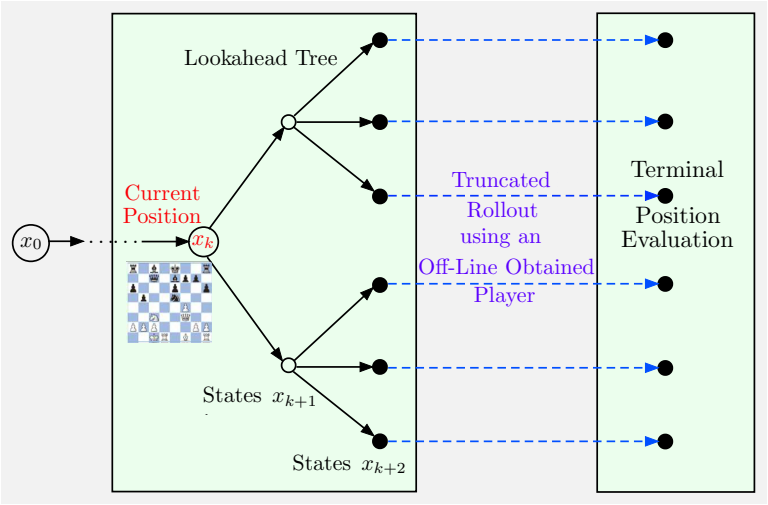


Figure 1.1.2 Illustration of an on-line player such as the one used in AlphaGo, AlphaZero, and Tesauro's backgammon program [TeG96]. At a given position, it generates a lookahead tree of multiple moves up to a given depth, then runs the off-line obtained player for some more moves, and then evaluates the effect of the remaining moves by using the position evaluator of the off-line obtained player.

with sophisticated off-line training. This phenomenon, *policy improvement through on-line play*, is centrally important for our purposes in this book.

Given the policy network/player obtained off-line and its value network/position evaluator, the on-line algorithm plays roughly as follows (see Fig. 1.1.2). At a given position, it generates a lookahead tree of all possible multiple move and countermove sequences, up to a given depth. It then runs the off-line obtained player for some more moves, and then evaluates the effect of the remaining moves by using the position evaluator of the value network. The middle portion, called "truncated rollout," may be viewed as *an economical substitute for longer lookahead*. Actually truncated rollout is not used in the published version of AlphaZero [SHS17]; the first portion (multistep lookahead) is quite long and implemented efficiently, so that the rollout portion is not essential. However, rollout is used in AlphaGo [SHM16]. Moreover, chess and Go programs (including AlphaZero) typically use a limited form of rollout, called "quiescence search," which aims to resolve imminent threats and highly dynamic positions before invoking the position evaluator. Rollout is instrumental in achieving high performance in Tesauro's 1996 backgammon program [TeG96]. The reason is that backgammon involves stochastic uncertainty, so long lookahead is not possible because of rapid expansion of the lookahead tree with every move.†

† Tesauro's rollout-based backgammon program [TeG96] uses only a value

We should note that the preceding description of AlphaZero and related games is oversimplified. We will be adding refinements and details as the book progresses. However, DP ideas with cost function approximations, similar to the on-line player illustrated in Fig. 1.1.2, will be central for our purposes. They will be generically referred to as *approximation in value space*. Moreover, the conceptual division between off-line training and on-line policy implementation will be important for our purposes.

Note also that these two processes may be decoupled and may be designed independently. For example the off-line training portion may be very simple, such as using a known heuristic policy for rollout without truncation, or without terminal cost approximation. Conversely, a sophisticated process may be used for off-line training of a terminal cost function approximation, which is used immediately following one-step or multistep lookahead in a value space approximation scheme.

1.2 DETERMINISTIC DYNAMIC PROGRAMMING

In all DP problems, the central object is a discrete-time dynamic system that generates a sequence of states under the influence of control. The system may evolve deterministically or randomly (under the additional influence of a random disturbance).

1.2.1 Finite Horizon Problem Formulation

In finite horizon problems the system evolves over a finite number N of time steps (also called stages). The state and control at time k of the system will be generally denoted by x_k and u_k, respectively. In deterministic systems, x_{k+1} is generated nonrandomly, i.e., it is determined solely by x_k and u_k. Thus, a deterministic DP problem involves a system of the form

$$x_{k+1} = f_k(x_k, u_k), \qquad k = 0, 1, \ldots, N-1, \tag{1.1}$$

where k is the time index, and

x_k is the state of the system, an element of some space,

u_k is the control or decision variable, to be selected at time k from some given set $U_k(x_k)$ that depends on x_k,

---

iteration scheme developed several years earlier [Tes94]. TD-Gammon is used to generate moves for the truncated rollout via a one-step or two-step lookahead minimization. Thus the value network also serves as a substitute for the policy network during the rollout operation. The terminal position evaluation used at the end of the truncated rollout is also provided by the value network. The middle portion of Tesauro's scheme (truncated rollout) is important for achieving a very high quality of play, as it effectively extends the length of lookahead from the current position.

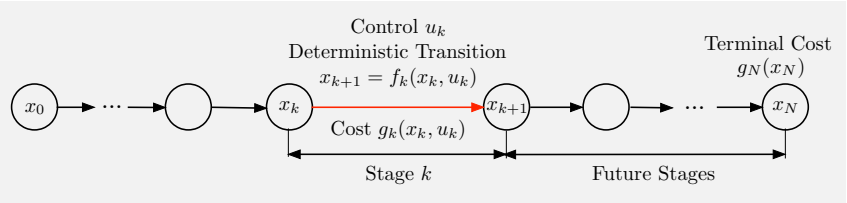


Figure 1.2.1 Illustration of a deterministic N-stage optimal control problem. Starting from state x_k, the next state under control u_k is generated nonrandomly, according to

$$x_{k+1} = f_k(x_k, u_k),$$

and a stage cost $g_k(x_k, u_k)$ is incurred.

f_k is a function of (x_k, u_k) that describes the mechanism by which the state is updated from time k to time $k+1$,

N is the horizon, i.e., the number of times control is applied.

The set of all possible x_k is called the *state space* at time k. It can be any set and may depend on k. Similarly, the set of all possible u_k is called the *control space* at time k. Again it can be any set and may depend on k. Similarly the system function f_k can be arbitrary and may depend on k.†

The problem also involves a cost function that is additive in the sense that the cost incurred at time k, denoted by $g_k(x_k, u_k)$, accumulates over time. Formally, g_k is a function of (x_k, u_k) that takes real number values, and may depend on k. For a given initial state x_0, the total cost of a control sequence $\{u_0, \ldots, u_{N-1}\}$ is

$$J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k), \tag{1.2}$$

† This generality is one of the great strengths of the DP methodology and guides the exposition style of this book, and the author's other DP works. By allowing arbitrary state and control spaces (discrete, continuous, or mixtures thereof), and a k-dependent choice of these spaces, we can focus attention on the truly essential algorithmic aspects of the DP approach, exclude extraneous assumptions and constraints from our model, and avoid duplication of analysis.

The generality of our DP model is also partly responsible for our choice of notation. In the artificial intelligence and operations research communities, finite state models, often referred to as Markovian Decision Problems (MDP), are common and use a transition probability notation (see Chapter 5). Unfortunately, this notation is not well suited for deterministic models, and also for continuous spaces models, both of which are important for the purposes of this book. For the latter models, it involves transition probability distributions over continuous spaces, and leads to mathematics that are far more complex as well as less intuitive than those based on the use of the system function (1.1).

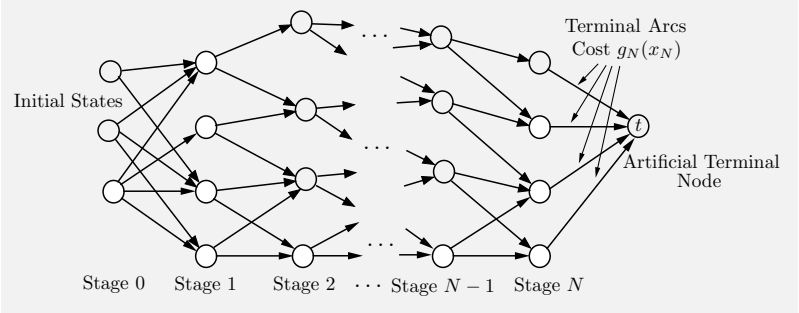


Figure 1.2.2 Transition graph for a deterministic finite-state system. Nodes correspond to states x_k. Arcs correspond to state-control pairs (x_k, u_k). An arc (x_k, u_k) has start and end nodes x_k and $x_{k+1} = f_k(x_k, u_k)$, respectively. The transition cost $g_k(x_k, u_k)$ is viewed as the length of this arc. The problem is equivalent to finding a shortest path from initial nodes of stage 0 to an artificial terminal node t.

where $g_N(x_N)$ is a terminal cost incurred at the end of the process. This is a well-defined number, since the control sequence $\{u_0, \ldots, u_{N-1}\}$ together with x_0 determines exactly the state sequence $\{x_1, \ldots, x_N\}$ via the system equation (1.1); see Figure 1.2.1. We want to minimize the cost (1.2) over all sequences $\{u_0, \ldots, u_{N-1}\}$ that satisfy the control constraints, thereby obtaining the optimal value as a function of x_0:†

$$J^*(x_0) = \min_{\substack{u_k \in U_k(x_k) \\ k=0,\ldots,N-1}} J(x_0; u_0, \ldots, u_{N-1}).$$

Discrete Optimal Control Problems

There are many situations where the state and control spaces are naturally discrete and consist of a finite number of elements. Such problems are often conveniently described with an acyclic graph specifying for each state x_k the possible transitions to next states x_{k+1}. The nodes of the graph correspond to states x_k and the arcs of the graph correspond to state-control pairs (x_k, u_k). Each arc with start node x_k corresponds to a choice of a single control $u_k \in U_k(x_k)$ and has as end node the next state $f_k(x_k, u_k)$. The cost of an arc (x_k, u_k) is defined as $g_k(x_k, u_k)$; see Fig. 1.2.2. To handle the final stage, an artificial terminal node t is added. Each state x_N at stage N is connected to the terminal node t with an arc having cost $g_N(x_N)$.

Note that control sequences $\{u_0, \ldots, u_{N-1}\}$ correspond to paths originating at the initial state (a node at stage 0) and terminating at one of the nodes corresponding to the final stage N. If we view the cost of an arc as

† Here and later we write "min" (rather than "inf") even if we are not sure that the minimum is attained. Similarly we write "max" (rather than "sup") even if we are not sure that the maximum is attained.

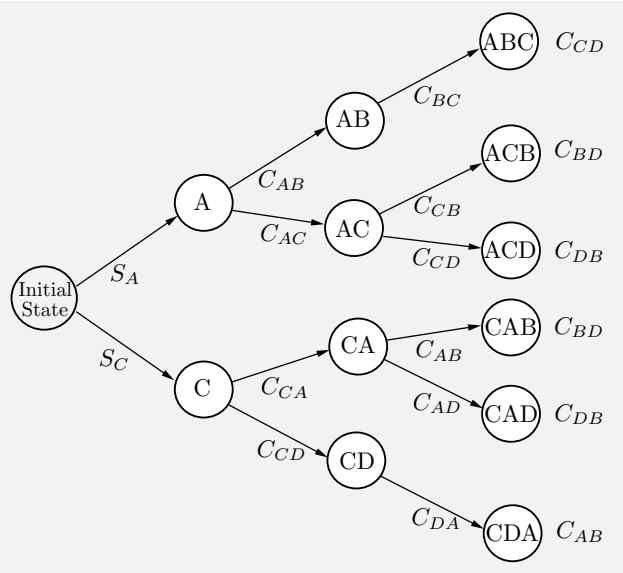


Figure 1.2.3 The transition graph of the deterministic scheduling problem of Example 1.2.1. Each arc of the graph corresponds to a decision leading from some state (the start node of the arc) to some other state (the end node of the arc). The corresponding cost is shown next to the arc. The cost of the last operation is shown as a terminal cost next to the terminal nodes of the graph.

its length, we see that *a deterministic finite-state finite-horizon problem is equivalent to finding a minimum-length (or shortest) path from the initial nodes of the graph (stage 0) to the terminal node t*. Here, by the length of a path we mean the sum of the lengths of its arcs.†

Generally, combinatorial optimization problems can be formulated as deterministic finite-state finite-horizon optimal control problem, as we will discuss in greater detail in Chapters 2 and 3. The idea is to break down the solution into components, which can be computed sequentially. The following is an illustrative example.

Example 1.2.1 (A Deterministic Scheduling Problem)

Suppose that to produce a certain product, four operations must be performed on a certain machine. The operations are denoted by A, B, C, and D. We assume that operation B can be performed only after operation A has been performed, and operation D can be performed only after operation C has been performed. (Thus the sequence CDAB is allowable but the sequence CDBA

† It turns out also that any shortest path problem (with a possibly nonacyclic graph) can be reformulated as a finite-state deterministic optimal control problem. See [Ber17a], Section 2.1, and [Ber91], [Ber98] for extensive accounts of shortest path methods, which connect with our discussion here.

is not.) The setup cost C_{mn} for passing from any operation m to any other operation n is given (cf. Fig. 1.2.3). There is also an initial startup cost S_A or S_C for starting with operation A or C, respectively. The cost of a sequence is the sum of the setup costs associated with it; for example, the operation sequence ACDB has cost $S_A + C_{AC} + C_{CD} + C_{DB}$.

We can view this problem as a sequence of three decisions, namely the choice of the first three operations to be performed (the last operation is determined from the preceding three). It is appropriate to consider as state the set of operations already performed, the initial state being an artificial state corresponding to the beginning of the decision process. The possible state transitions corresponding to the possible states and decisions for this problem are shown in Fig. 1.2.3. Here the problem is deterministic, i.e., at a given state, each choice of control leads to a uniquely determined state. For example, at state AC the decision to perform operation D leads to state ACD with certainty, and has cost C_{CD}. Thus the problem can be conveniently represented with the transition graph of Fig. 1.2.3. The optimal solution corresponds to the path that starts at the initial state and ends at some state at the terminal time and has minimum sum of arc costs plus the terminal cost.

1.2.2 The Dynamic Programming Algorithm

In this section we will state the DP algorithm and formally justify it. The algorithm rests on a simple idea, the *principle of optimality*, which roughly states the following; see Fig. 1.2.4.

Principle of Optimality

Let $\{u_0^*, \ldots, u_{N-1}^*\}$ be an optimal control sequence, which together with x_0 determines the corresponding state sequence $\{x_1^*, \ldots, x_N^*\}$ via the system equation (1.1). Consider the subproblem whereby we start at x_k^* at time k and wish to minimize the "cost-to-go" from time k to time N,

$$g_k(x_k^*, u_k) + \sum_{m=k+1}^{N-1} g_m(x_m, u_m) + g_N(x_N),$$

over $\{u_k, \ldots, u_{N-1}\}$ with $u_m \in U_m(x_m)$, $m = k, \ldots, N-1$. Then the truncated optimal control sequence $\{u_k^*, \ldots, u_{N-1}^*\}$ is optimal for this subproblem.

The subproblem referred to above is called the *tail subproblem* that starts at x_k^*. Stated succinctly, the principle of optimality says that *the tail of an optimal sequence is optimal for the tail subproblem*. Its intuitive justification is simple. If the truncated control sequence $\{u_k^*, \ldots, u_{N-1}^*\}$ were not optimal as stated, we would be able to reduce the cost further by switching to an optimal sequence for the subproblem once we reach x_k^*

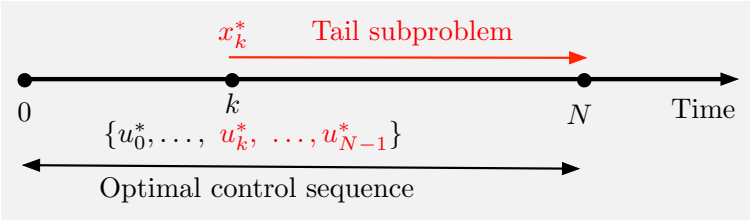


Figure 1.2.4 Schematic illustration of the principle of optimality. The tail $\{u_k^*, \ldots, u_{N-1}^*\}$ of an optimal sequence $\{u_0^*, \ldots, u_{N-1}^*\}$ is optimal for the tail subproblem that starts at the state x_k^* of the optimal state trajectory.

(since the preceding choices of controls, $u_0^*, \ldots, u_{k-1}^*$, do not restrict our future choices).

For an auto travel analogy, suppose that the fastest route from Phoenix to Boston passes through St Louis. The principle of optimality translates to the obvious fact that the St Louis to Boston portion of the route is also the fastest route for a trip that starts from St Louis and ends in Boston.†

The principle of optimality suggests that the optimal cost function can be constructed in piecemeal fashion going backwards: first compute the optimal cost function for the "tail subproblem" involving the last stage, then solve the "tail subproblem" involving the last two stages, and continue in this manner until the optimal cost function for the entire problem is constructed.

The DP algorithm is based on this idea: it proceeds sequentially, by *solving all the tail subproblems of a given time length, using the solution of the tail subproblems of shorter time length*. We illustrate the algorithm with the scheduling problem of Example 1.2.1. The calculations are simple but tedious, and may be skipped without loss of continuity. However, they may be worth going over by a reader that has no prior experience in the use of DP.

Example 1.2.1 (Scheduling Problem - Continued)

Let us consider the scheduling Example 1.2.1, and let us apply the principle of optimality to calculate the optimal schedule. We have to schedule optimally the four operations A, B, C, and D. There is a cost for a transition between two operations, and the numerical values of the transition costs are shown in Fig. 1.2.5 next to the corresponding arcs.

According to the principle of optimality, the "tail" portion of an optimal schedule must be optimal. For example, suppose that the optimal schedule

† In the words of Bellman [Bel57]: "An optimal trajectory has the property that at an intermediate point, no matter how it was reached, the rest of the trajectory must coincide with an optimal trajectory as computed from this intermediate point as the starting point."

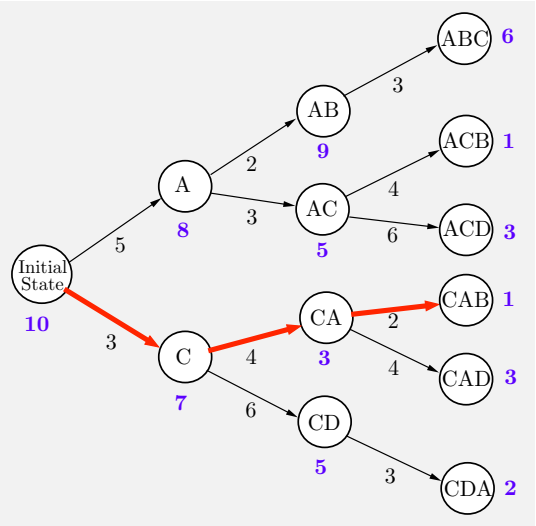


Figure 1.2.5 Transition graph of the deterministic scheduling problem, with the cost of each decision shown next to the corresponding arc. Next to each node/state we show the cost to optimally complete the schedule starting from that state. This is the optimal cost of the corresponding tail subproblem (cf. the principle of optimality). The optimal cost for the original problem is equal to 10, as shown next to the initial state. The optimal schedule corresponds to the thick-line arcs.

is CABD. Then, having scheduled first C and then A, it must be optimal to complete the schedule with BD rather than with DB. With this in mind, we solve all possible tail subproblems of length two, then all tail subproblems of length three, and finally the original problem that has length four (the subproblems of length one are of course trivial because there is only one operation that is as yet unscheduled). As we will see shortly, the tail subproblems of length $k + 1$ are easily solved once we have solved the tail subproblems of length k, and this is the essence of the DP technique.

Tail Subproblems of Length 2: These subproblems are the ones that involve two unscheduled operations and correspond to the states AB, AC, CA, and CD (see Fig. 1.2.5).

State AB: Here it is only possible to schedule operation C as the next operation, so the optimal cost of this subproblem is 9 (the cost of scheduling C after B, which is 3, plus the cost of scheduling D after C, which is 6).

State AC: Here the possibilities are to (a) schedule operation B and then D, which has cost 5, or (b) schedule operation D and then B, which has cost 9. The first possibility is optimal, and the corresponding cost of the tail subproblem is 5, as shown next to node AC in Fig. 1.2.5.

State CA: Here the possibilities are to (a) schedule operation B and then

D, which has cost 3, or (b) schedule operation D and then B, which has cost 7. The first possibility is optimal, and the corresponding cost of the tail subproblem is 3, as shown next to node CA in Fig. 1.2.5.

State CD: Here it is only possible to schedule operation A as the next operation, so the optimal cost of this subproblem is 5.

Tail Subproblems of Length 3: These subproblems can now be solved using the optimal costs of the subproblems of length 2.

State A: Here the possibilities are to (a) schedule next operation B (cost 2) and then solve optimally the corresponding subproblem of length 2 (cost 9, as computed earlier), a total cost of 11, or (b) schedule next operation C (cost 3) and then solve optimally the corresponding subproblem of length 2 (cost 5, as computed earlier), a total cost of 8. The second possibility is optimal, and the corresponding cost of the tail subproblem is 8, as shown next to node A in Fig. 1.2.5.

State C: Here the possibilities are to (a) schedule next operation A (cost 4) and then solve optimally the corresponding subproblem of length 2 (cost 3, as computed earlier), a total cost of 7, or (b) schedule next operation D (cost 6) and then solve optimally the corresponding subproblem of length 2 (cost 5, as computed earlier), a total cost of 11. The first possibility is optimal, and the corresponding cost of the tail subproblem is 7, as shown next to node C in Fig. 1.2.5.

Original Problem of Length 4: The possibilities here are (a) start with operation A (cost 5) and then solve optimally the corresponding subproblem of length 3 (cost 8, as computed earlier), a total cost of 13, or (b) start with operation C (cost 3) and then solve optimally the corresponding subproblem of length 3 (cost 7, as computed earlier), a total cost of 10. The second possibility is optimal, and the corresponding optimal cost is 10, as shown next to the initial state node in Fig. 1.2.5.

Note that having computed the optimal cost of the original problem through the solution of all the tail subproblems, we can construct the optimal schedule: we begin at the initial node and proceed forward, each time choosing the optimal operation, i.e., the one that starts the optimal schedule for the corresponding tail subproblem. In this way, by inspection of the graph and the computational results of Fig. 1.2.5, we determine that CABD is the optimal schedule.

Finding an Optimal Control Sequence by DP

We now state the DP algorithm for deterministic finite horizon problems by translating into mathematical terms the heuristic argument underlying the principle of optimality. The algorithm constructs functions

$$J_N^*(x_N), J_{N-1}^*(x_{N-1}), \ldots, J_0^*(x_0),$$

sequentially, starting from J_N^*, and proceeding backwards to J_{N-1}^*, J_{N-2}^*, etc. *The value $J_k^*(x_k)$ represents the optimal cost of the tail subproblem that starts at state x_k at time k.*

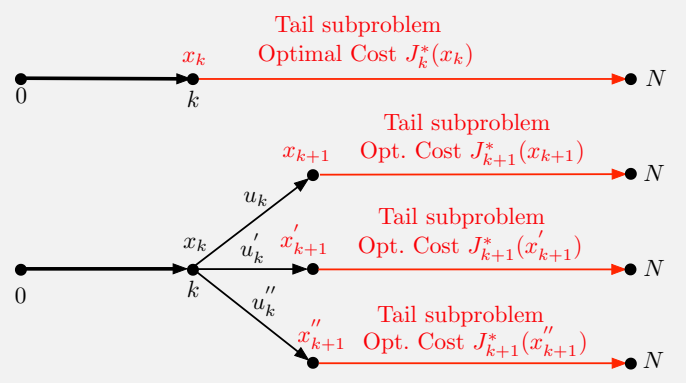


Figure 1.2.6 Illustration of the DP algorithm. The tail subproblem that starts at x_k at time k minimizes over $\{u_k, \ldots, u_{N-1}\}$ the "cost-to-go" from k to N,

$$g_k(x_k, u_k) + \sum_{m=k+1}^{N-1} g_m(x_m, u_m) + g_N(x_N).$$

To solve it, we choose u_k to minimize the (1st stage cost + Optimal tail problem cost) or

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big) \Big].$$

DP Algorithm for Deterministic Finite Horizon Problems

Start with

$$J_N^*(x_N) = g_N(x_N), \qquad \text{for all } x_N, \tag{1.3}$$

and for $k = 0, \ldots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big) \Big], \qquad \text{for all } x_k. \tag{1.4}$$

The DP algorithm together with the construction of the optimal cost-to-go functions $J_k^*(x_k)$ are illustrated in Fig. 1.2.6. Note that at stage k, the calculation in Eq. (1.4) must be done for all states x_k before proceeding to stage $k-1$. The key fact about the DP algorithm is that for every initial state x_0, the number $J_0^*(x_0)$ obtained at the last step, is equal to the optimal cost $J^*(x_0)$. Indeed, a more general fact can be shown, namely that for all $k = 0, 1, \ldots, N-1$, and all states x_k at time k, we have

$$J_k^*(x_k) = \min_{\substack{u_m \in U_m(x_m) \\ m=k,\ldots,N-1}} J(x_k; u_k, \ldots, u_{N-1}), \tag{1.5}$$

where $J(x_k; u_k, \ldots, u_{N-1})$ is the cost generated by starting at x_k and using subsequent controls $u_k, \ldots, u_{N-1}$:

$$J(x_k; u_k, \ldots, u_{N-1}) = g_N(x_N) + \sum_{t=k}^{N-1} g_t(x_t, u_t). \tag{1.6}$$

Thus, $J_k^*(x_k)$ is the optimal cost for an $(N-k)$-stage tail subproblem that starts at state x_k and time k, and ends at time N.† Based on the interpretation (1.5) of $J_k^*(x_k)$, we call it the *optimal cost-to-go* from state x_k at stage k, and refer to J_k^* as the *optimal cost-to-go function* or *optimal cost function* at time k. In maximization problems the DP algorithm (1.4) is written with maximization in place of minimization, and then J_k^* is referred to as the *optimal value function* at time k.

Once the functions $J_0^*, \ldots, J_N^*$ have been obtained, we can use a forward algorithm to construct an optimal control sequence $\{u_0^*, \ldots, u_{N-1}^*\}$ and state trajectory $\{x_1^*, \ldots, x_N^*\}$ for the given initial state x_0.

Construction of Optimal Control Sequence $\{u_0^*, \ldots, u_{N-1}^*\}$

Set

$$u_0^* \in \arg \min_{u_0 \in U_0(x_0)} \Big[g_0(x_0, u_0) + J_1^*\big(f_0(x_0, u_0)\big) \Big],$$

and

$$x_1^* = f_0(x_0, u_0^*).$$

Sequentially, going forward, for $k = 1, 2, \ldots, N-1$, set

† We can prove this by induction. The assertion holds for $k = N$ in view of the initial condition $J_N^*(x_N) = g_N(x_N)$. To show that it holds for all k, we use Eqs. (1.5) and (1.6) to write

$$\begin{aligned} J_k^*(x_k) &= \min_{\substack{u_t \in U_t(x_t) \\ t=k,\ldots,N-1}} \left[g_N(x_N) + \sum_{t=k}^{N-1} g_t(x_t, u_t) \right] \\ &= \min_{u_k \in U_k(x_k)} \left[g_k(x_k, u_k) + \min_{\substack{u_t \in U_t(x_t) \\ t=k+1,\ldots,N-1}} \left[g_N(x_N) + \sum_{t=k+1}^{N-1} g_t(x_t, u_t) \right] \right] \\ &= \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big) \Big], \end{aligned}$$

where for the last equality we use the induction hypothesis. A subtle mathematical point here is that, through the minimization operation, the functions J_k^* may take the value $-\infty$ for some x_k. Still the preceding induction argument is valid even if this is so. The books [BeT96] and [Ber18a] address DP algorithms that allow infinite values in various operations such as minimization.

$$u_k^* \in \arg \min_{u_k \in U_k(x_k^*)} \Big[g_k(x_k^*, u_k) + J_{k+1}^*\big(f_k(x_k^*, u_k)\big) \Big], \qquad (1.7)$$

and

$$x_{k+1}^* = f_k(x_k^*, u_k^*).$$

Note an interesting conceptual division of the optimal control sequence construction: there is "off-line training" to obtain J_k^* by precomputation [cf. Eqs. (1.3)-(1.4)], which is followed by real-time "on-line play" to obtain u_k^* [cf. Eq. (1.7)]. This is analogous to the two algorithmic processes described in Section 1.1 in connection with chess and backgammon.

Figure 1.2.5 traces the calculations of the DP algorithm for the scheduling Example 1.2.1. The numbers next to the nodes, give the corresponding cost-to-go values, and the thick-line arcs give the construction of the optimal control sequence using the preceding algorithm.

DP Algorithm for General Discrete Optimization Problems

We have noted earlier that discrete deterministic optimization problems, including challenging combinatorial problems, can be typically formulated as DP problems by breaking down each feasible solution into a sequence of decisions/controls, as illustrated with the scheduling Example 1.2.1. This formulation often leads to an intractable DP computation because of an exponential explosion of the number of states as time progresses. However, a DP formulation brings to bear approximate DP methods, such as rollout and others, to be discussed shortly, which can deal with the exponentially increasing size of the state space. We illustrate the reformulation by an example and then generalize.

Example 1.2.2 (The Traveling Salesman Problem)

An important model for scheduling a sequence of operations is the classical traveling salesman problem. Here we are given N cities and the travel time between each pair of cities. We wish to find a minimum time travel that visits each of the cities exactly once and returns to the start city. To convert this problem to a DP problem, we form a graph whose nodes are the sequences of k distinct cities, where $k = 1, \ldots, N$. The k-city sequences correspond to the states of the kth stage. The initial state x_0 consists of some city, taken as the start (city A in the example of Fig. 1.2.7). A k-city node/state leads to a $(k+1)$-city node/state by adding a new city at a cost equal to the travel time between the last two of the $k+1$ cities; see Fig. 1.2.7. Each sequence of N cities is connected to an artificial terminal node t with an arc of cost equal to the travel time from the last city of the sequence to the starting city, thus completing the transformation to a DP problem.

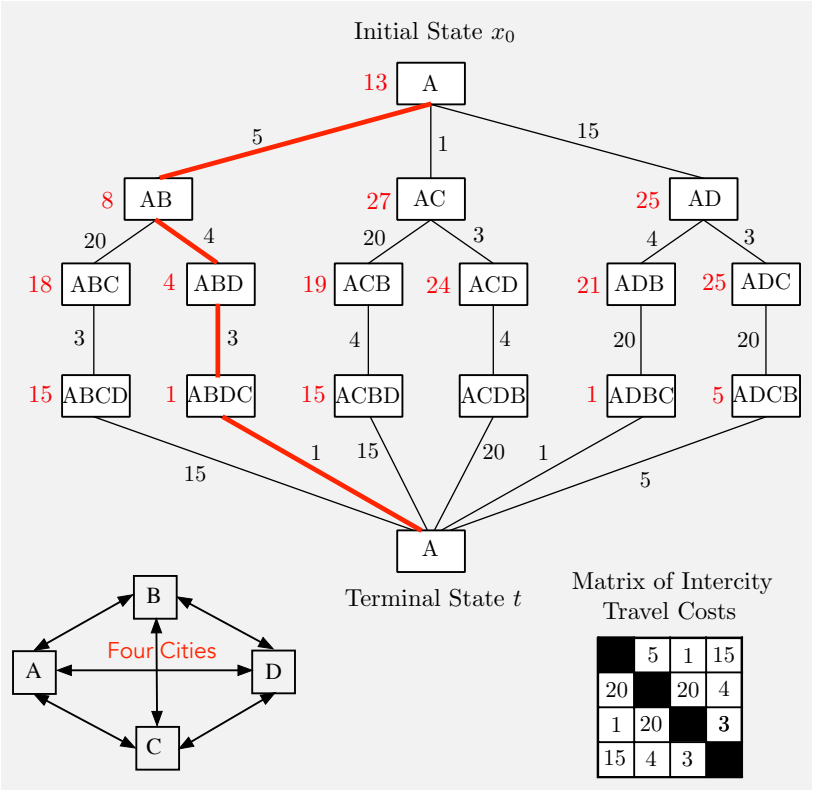


Figure 1.2.7 Example of a DP formulation of the traveling salesman problem. The travel times between the four cities A, B, C, and D are shown in the matrix at the bottom. We form a graph whose nodes are the k-city sequences and correspond to the states of the kth stage, assuming that A is the starting city. The transition costs/travel times are shown next to the arcs. The optimal costs-to-go are generated by DP starting from the terminal state and going backwards towards the initial state, and are shown next to the nodes. There is a unique optimal sequence here (ABDCA), and it is marked with thick lines. The optimal sequence can be obtained by forward minimization [cf. Eq. (1.7)], starting from the initial state x_0.

The optimal costs-to-go from each node to the terminal state can be obtained by the DP algorithm and are shown next to the nodes. Note, however, that the number of nodes grows exponentially with the number of cities N. This makes the DP solution intractable for large N. As a result, large traveling salesman and related scheduling problems are typically addressed with approximation methods, some of which are based on DP, and will be discussed in future chapters.

Let us now extend the ideas of the preceding example to the general

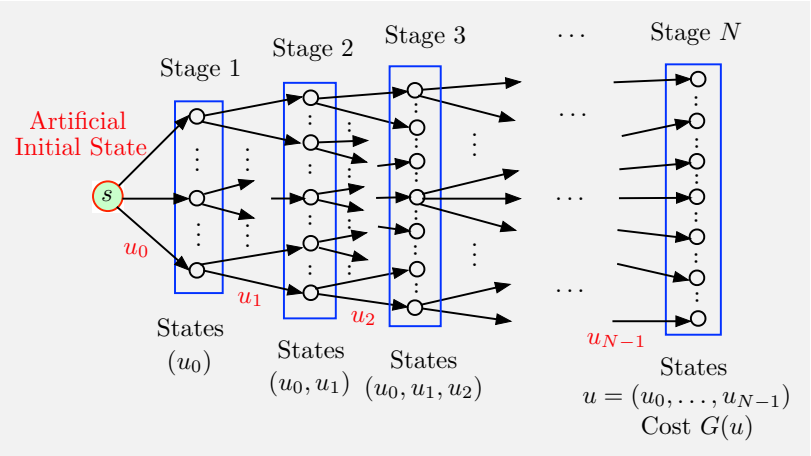


Figure 1.2.8 Formulation of a discrete optimization problem as a DP problem with N stages. There is a cost $G(u)$ only at the terminal stage on the arc connecting an N-solution $u = (u_0, \ldots, u_{N-1})$ upon reaching the terminal state. Note that there is only one incoming arc at each node.

discrete optimization problem:

$$\begin{aligned} &\text{minimize} \quad G(u) \\ &\text{subject to} \quad u \in U, \end{aligned}$$

where U is a finite set of feasible solutions and $G(u)$ is a cost function. We assume that each solution u has N components; i.e., it has the form $u = (u_0, \ldots, u_{N-1})$, where N is a positive integer. We can then view the problem as a sequential decision problem, where the components $u_0, \ldots, u_{N-1}$ are selected one-at-a-time. A k-tuple $(u_0, \ldots, u_{k-1})$ consisting of the first k components of a solution is called a *k-solution*. We associate k-solutions with the kth stage of the finite horizon DP problem shown in Fig. 1.2.8. In particular, for $k = 1, \ldots, N$, we view as the states of the kth stage all the k-tuples $(u_0, \ldots, u_{k-1})$. For stage $k = 0, \ldots, N-1$, we view u_k as the control. The initial state is an artificial state denoted s. From this state, by applying u_0, we may move to any "state" (u_0), with u_0 belonging to the set

$$U_0 = \big\{\tilde{u}_0 \mid \text{there exists a solution of the form } (\tilde{u}_0, \tilde{u}_1, \ldots, \tilde{u}_{N-1}) \in U\big\}.$$

Thus U_0 is the set of choices of u_0 that are consistent with feasibility.

More generally, from a state $(u_0, \ldots, u_{k-1})$, we may move to any state of the form $(u_0, \ldots, u_{k-1}, u_k)$, upon choosing a control u_k that belongs to the set

$$U_k(u_0, \ldots, u_{k-1}) = \big\{u_k \mid \text{for some } \overline{u}_{k+1}, \ldots, \overline{u}_{N-1} \text{ we have } (u_0, \ldots, u_{k-1}, u_k, \overline{u}_{k+1}, \ldots, \overline{u}_{N-1}) \in U\big\}.$$

These are the choices of u_k that are consistent with the preceding choices $u_0, \ldots, u_{k-1}$, and are also consistent with feasibility. The last stage corresponds to the N-solutions $u = (u_0, \ldots, u_{N-1})$, and the terminal cost is $G(u)$; see Fig. 1.2.8. All other transitions in this DP problem formulation have cost 0.

Let

$$J_k^*(u_0, \ldots, u_{k-1})$$

denote the optimal cost starting from the k-solution $(u_0, \ldots, u_{k-1})$, i.e., the optimal cost of the problem over solutions whose first k components are constrained to be equal to $u_0, \ldots, u_{k-1}$. The DP algorithm is described by the equation

$$J_k^*(u_0, \ldots, u_{k-1}) = \min_{u_k \in U_k(u_0, \ldots, u_{k-1})} J_{k+1}^*(u_0, \ldots, u_{k-1}, u_k),$$

with the terminal condition

$$J_N^*(u_0, \ldots, u_{N-1}) = G(u_0, \ldots, u_{N-1}).$$

This algorithm executes backwards in time: starting with the known function $J_N^* = G$, we compute J_{N-1}^*, then J_{N-2}^*, and so on up to computing J_0^*. An optimal solution $(u_0^*, \ldots, u_{N-1}^*)$ is then constructed by going forward through the algorithm

$$u_k^* \in \arg \min_{u_k \in U_k(u_0^*, \ldots, u_{k-1}^*)} J_{k+1}^*(u_0^*, \ldots, u_{k-1}^*, u_k), \quad k = 0, \ldots, N-1, \quad (1.8)$$

first compute u_0^*, then u_1^*, and so on up to u_{N-1}^*; cf. Eq. (1.7).

Of course here the number of states typically grows exponentially with N, but we can use the DP minimization (1.8) as a starting point for the use of approximation methods. For example we may try to use approximation in value space, whereby we replace J_{k+1}^* with some suboptimal $\tilde{J}_{k+1}$ in Eq. (1.8). One possibility is to use as

$$\tilde{J}_{k+1}(u_0^*, \ldots, u_{k-1}^*, u_k),$$

the cost generated by a heuristic method that solves the problem suboptimally with the values of the first $k+1$ decision components fixed at $u_0^*, \ldots, u_{k-1}^*, u_k$. This is the *rollout algorithm*, which is a very simple and effective approach for approximate combinatorial optimization. It will be discussed in the next section, and in Chapters 2 and 3. It will be related to the method of policy iteration and self-learning ideas in Chapter 5.

Let us finally note that while we have used a general cost function G and constraint set C in our discrete optimization model of this section, in many problems G and/or C may have a special structure, which is consistent with a sequential decision making process. The traveling salesman Example 1.2.2 is a case in point, where G consists of N components (the intercity travel costs), one per stage.

1.2.3 Approximation in Value Space

The forward optimal control sequence construction of Eq. (1.7) is possible only after we have computed $J_k^*(x_k)$ by DP for all x_k and k. Unfortunately, in practice this is often prohibitively time-consuming, because the number of possible x_k and k can be very large. However, a similar forward algorithmic process can be used if the optimal cost-to-go functions J_k^* are replaced by some approximations $\tilde{J}_k$. This is the basis for an idea that is central in RL: *approximation in value space*.† It constructs a suboptimal solution $\{\tilde{u}_0, \ldots, \tilde{u}_{N-1}\}$ in place of the optimal $\{u_0^*, \ldots, u_{N-1}^*\}$, based on using $\tilde{J}_k$ in place of J_k^* in the DP procedure (1.7).

Approximation in Value Space - Use of $\tilde{J}_k$ in Place of J_k^*

Start with

$$\tilde{u}_0 \in \arg\min_{u_0 \in U_0(x_0)} \Big[g_0(x_0, u_0) + \tilde{J}_1\big(f_0(x_0, u_0)\big) \Big],$$

and set

$$\tilde{x}_1 = f_0(x_0, \tilde{u}_0).$$

Sequentially, going forward, for $k = 1, 2, \ldots, N-1$, set

$$\tilde{u}_k \in \arg\min_{u_k \in U_k(\tilde{x}_k)} \Big[g_k(\tilde{x}_k, u_k) + \tilde{J}_{k+1}\big(f_k(\tilde{x}_k, u_k)\big) \Big], \qquad (1.9)$$

and

$$\tilde{x}_{k+1} = f_k(\tilde{x}_k, \tilde{u}_k).$$

Thus in approximation in value space the calculation of the suboptimal sequence $\{\tilde{u}_0, \ldots, \tilde{u}_{N-1}\}$ is done by going forward (no backward calculation is needed once the approximate cost-to-go functions $\tilde{J}_k$ are available). This is similar to the calculation of the optimal sequence $\{u_0^*, \ldots, u_{N-1}^*\}$ [cf. Eq. (1.7)], and is independent of how the functions $\tilde{J}_k$ are computed.

Multistep Lookahead

The algorithm (1.9) is said to involve a *one-step lookahead minimization*, since it solves a one-stage DP problem for each k. In the next chapter we

† Approximation in value space is a simple idea that has been used quite extensively for deterministic problems, well before the development of the modern RL methodology. For example it underlies the widely used A^* method for computing approximate solutions to large scale shortest path problems.

will also discuss the possibility of *multistep lookahead*, which involves the solution of an ℓ-step DP problem, where ℓ is an integer, $1 < \ell < N-k$, with a terminal cost function approximation $\tilde{J}_{k+\ell}$. Multistep lookahead typically (but not always) provides better performance over one-step lookahead in RL approximation schemes, and will be discussed in Chapter 2. For example in Alphazero chess, long multistep lookahead is critical for good on-line performance. The intuitive reason is that with ℓ stages being treated "exactly" (by optimization), the effect of the approximation error

$$\tilde{J}_{k+\ell} - J^*_{k+\ell}$$

tends to become less significant as ℓ increases. However, the solution of the multistep lookahead optimization problem, instead of the one-step lookahead counterpart of Eq. (1.9), becomes more time consuming.

Rollout, Cost Improvement, and On-Line Replanning

A major issue in value space approximation is the construction of suitable approximate cost-to-go functions $\tilde{J}_k$. This can be done in many different ways, giving rise to some of the principal RL methods. For example, $\tilde{J}_k$ may be constructed with a sophisticated off-line training method, as discussed in Section 1.1, in connection with chess and backgammon. Alternatively, $\tilde{J}_k$ may be obtained on-line with *rollout*, which will be discussed in detail in this book, starting with the next chapter. In rollout, the approximate values $\tilde{J}_k(x_k)$ are obtained when needed by running a heuristic control scheme, called *base heuristic* or *base policy*, for a suitably large number of steps, starting from the state x_k.

The major theoretical property of rollout is *cost improvement*: the cost obtained by rollout using some base heuristic is less or equal to the corresponding cost of the base heuristic. This is true for any starting state, provided the base heuristic satisfies some simple conditions, which will be discussed in Chapter 2.†

There are also several variants of rollout, including versions involving multiple heuristics, combinations with other forms of approximation in value space methods, and multistep lookahead, which will be discussed in subsequent chapters. An important methodology that is closely related to

† For an intuitive justification of the cost improvement mechanism, note that the rollout control $\tilde{u}_k$ is calculated from Eq. (1.9) to attain the minimum over u_k over the sum of two terms: the first stage cost $g_k(\tilde{x}_k, u_k)$ plus the cost of the remaining stages ($k+1$ to N) using the heuristic controls. Thus rollout involves a first stage optimization (rather than just using the base heuristic), which accounts for the cost improvement. This reasoning also helps to explain why multistep lookahead tends to provide better performance than one-step lookahead in approximation in value space and rollout schemes.

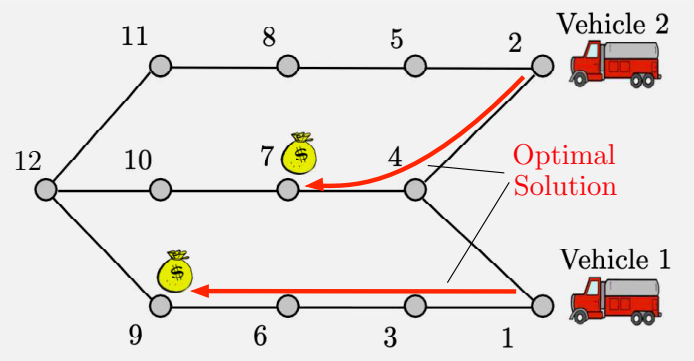


Figure 1.2.9 An instance of the vehicle routing problem of Example 1.2.3. The two vehicles aim to collectively perform the two tasks, at nodes 7 and 9, as fast as possible, by each moving to a neighboring node at each step. The optimal routes are shown and require a total of 5 vehicle moves.

deterministic rollout is *model predictive control*, which is used widely in control system design, and will be discussed in some detail in this book, starting with Section 3.1; see also Section 1.4.7. The following example is typical of the combinatorial applications that we will discuss in Chapter 3 in connection with rollout.

Example 1.2.3 (Multi-Vehicle Routing)

Consider n vehicles that move along the arcs of a given graph. Some of the nodes of the graph include a task to be performed by the vehicles. Each task will be performed only once, immediately after a vehicle reaches the corresponding node for the first time. We assume a horizon that is large enough to allow every task to be performed. The problem is to find a route for each vehicle so that, roughly speaking, the tasks are collectively performed by the vehicles in minimum time. To express this objective, we assume that for each move by a vehicle there is a cost of one unit. These costs are summed up to the point where all the tasks have been performed.

For a large number of vehicles and a complicated graph, this is a non-trivial combinatorial problem. It can be approached by DP, like any discrete deterministic optimization problem, as we discussed earlier, in Section 1.2.1. In particular, we can view as state at a given stage the n-tuple of current positions of the vehicles together with the list of pending tasks. Unfortunately, however, the number of these states can be enormous (it increases exponentially with the number of tasks and the number of vehicles), so an exact DP solution is intractable.

This motivates an optimization in value space approach based on rollout. For this we need an easily implementable base heuristic that will solve suboptimally the problem starting from any state x_{k+1}, and will provide the cost approximation $\tilde{J}_{k+1}(x_{k+1})$ in Eq. (1.9). One possibility is based on the vehicles choosing their actions selfishly, along shortest paths to their nearest

pending task, one-at-a-time in a fixed order. To illustrate, consider the two-vehicle problem of Fig. 1.2.9. We introduce a vehicle order (say, first vehicle 1 then vehicle 2), and move each vehicle in turn one step to the next node towards its nearest pending task, until all tasks have been performed.

The rollout algorithm will work as follows. At a given state x_k [involving for example vehicle positions at the node pair $(1, 2)$ and tasks at nodes 7 and 9, as in Fig. 1.2.9], we consider all possible joint vehicle moves (the controls u_k at the state) resulting in the node pairs (3,5), (4,5), (3,4) (4,4), corresponding to the next states x_{k+1} [thus, as an example (3,5) corresponds to vehicle 1 moving from 1 to 3, and vehicle 2 moving from 2 to 5]. We then run the heuristic starting from each of these node pairs, and accumulate the incurred costs up to the time when both tasks are completed. For example starting from the vehicle positions/next state (3,5), the heuristic will produce the following sequence of moves:

- Vehicle 1 moves from 3 to 6.
- Vehicle 2 moves from 5 to 2.
- Vehicle 1 moves from 6 to 9 and performs the task at 9.
- Vehicle 2 moves from 2 to 4.
- Vehicle 1 moves from 9 to 12.
- Vehicle 2 moves from 4 to 7 and performs the task at 7.

The two tasks are thus performed in 6 moves once the move to (3,5) has been made.

The process of running the heuristic is repeated from the other three vehicle position pairs/next states (4,5), (3,4) (4,4), and the heuristic cost (number of moves) is recorded. We then choose the next state that corresponds to minimum cost. In our case the joint move to state x_{k+1} that involves the pair $(3, 4)$ produces the sequence

- Vehicle 1 moves from 3 to 6,
- Vehicle 2 moves from 4 to 7 and performs the task at 7,
- Vehicle 1 moves from 6 to 9 and performs the task at 9,

and performs the two tasks in 3 moves. It can be seen that it yields minimum first stage cost plus heuristic cost from the next state, as per Eq. (1.9). Thus, the rollout algorithm will choose it at the state (1,2), and move the vehicles to state (3,4). At that state the rollout process will be repeated, i.e., consider the possible next joint moves to the node pairs (6,7), (6,2), (6,1), (3,7), (3,2), (3,1), perform a heuristic calculation from each of them, compare, etc.

It can be verified that the rollout algorithm starting from the state (1,2) shown in Fig. 1.2.9 will attain the optimal cost (a total of 5 vehicle moves). It will perform much better than the heuristic, which starting from state (1,2), will move the two vehicles to state (4,4) and then to (7,1), etc (a total of 9 vehicle moves). This is an instance of the cost improvement property of the rollout algorithm: it performs better than its base heuristic.

A related context where rollout can be very useful is when an optimal

solution has been derived for a given mathematical or simulation-based model of the problem, and the model changes as the system is operating. Then the solution at hand is not optimal anymore, but it can be used as a base heuristic for rollout using the new model, thereby restoring much of the optimality loss due to the change in model. This is known as *on-line replanning with rollout*, and will be discussed further in Section 1.4.7. For example, in the preceding multi-vehicle example, on-line replanning would perform well if the number and location of tasks may change unpredictably, as new tasks appear or old tasks disappear, or if some of the vehicles break down and get repaired randomly over time.

Lookahead Simplification and Multiagent Problems

Regardless of the method used to select the approximate cost-to-go functions $\tilde{J}_k$, another important issue is to perform efficiently the minimization over $u_k \in U_k(\tilde{x}_k)$ in Eq. (1.9). This minimization can be very time-consuming or even impossible, in which case it must be simplified for practical use.

An important example is when the control consists of multiple components, $u_k = (u_k^1, \ldots, u_k^m)$, with each component taking values in a finite set. Then the size of the control space grows exponentially with m, and may make the minimization intractable even for relatively small values of m. This situation arises in several types of problems, including the case of *multiagent problems*, where each control component is chosen by a separate decision maker, who will be referred to as an "agent." For instance in the multi-vehicle routing Example 1.2.3, each vehicle may be viewed as an agent, and the number of joint move choices by the vehicles increases exponentially with their number. This motivates another rollout approach, called *multiagent* or *agent-by-agent rollout*, which we will discuss in Section 1.4.5.

Another case where the lookahead minimization in Eq. (1.9) may become very difficult is when u_k takes a continuum of values. Then, it is necessary to either approximate the control constraint set $U_k(\tilde{x}_k)$ by discretization or sampling, or to use a continuous optimization algorithm such as a gradient or Newton-like method. A coordinate descent-type method may also be used in the multiagent case where the control consists of multiple components, $u_k = (u_k^1, \ldots, u_k^m)$. These possibilities will be considered further in subsequent chapters. An important case in point is the model predictive control methodology, which will be illustrated in Section 1.4.7 and discussed extensively later, starting with Section 3.1.

Q-Factors and Q-Learning

An alternative (and equivalent) form of the DP algorithm (1.4), uses the optimal cost-to-go functions J_k^* indirectly. In particular, it generates the

optimal Q-factors, defined for all pairs (x_k, u_k) and k by

$$Q_k^*(x_k, u_k) = g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big). \tag{1.10}$$

Thus the optimal Q-factors are simply the expressions that are minimized in the right-hand side of the DP equation (1.4).

Note that the optimal cost function J_k^* can be recovered from the optimal Q-factor Q_k^* by means of the minimization

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} Q_k^*(x_k, u_k). \tag{1.11}$$

Moreover, the DP algorithm (1.4) can be written in an essentially equivalent form that involves Q-factors only [cf. Eqs. (1.10)-(1.11)]:

$$Q_k^*(x_k, u_k) = g_k(x_k, u_k) + \min_{u_{k+1} \in U_{k+1}(f_k(x_k, u_k))} Q_{k+1}^*\big(f_k(x_k, u_k), u_{k+1}\big).$$

Exact and approximate forms of this and other related algorithms, including counterparts for stochastic optimal control problems, comprise an important class of RL methods known as *Q-learning*.

The expression

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k)\big),$$

which is minimized in approximation in value space [cf. Eq. (1.9)] is known as the (approximate) *Q-factor of* (x_k, u_k).† Note that the computation of the suboptimal control (1.9) can be done through the Q-factor minimization

$$\tilde{u}_k \in \arg\min_{u_k \in U_k(\tilde{x}_k)} \tilde{Q}_k(\tilde{x}_k, u_k).$$

This suggests the possibility of using Q-factors in place of cost functions in approximation in value space schemes. We will discuss such schemes later.

1.3 STOCHASTIC DYNAMIC PROGRAMMING

We will now extend the DP algorithm and our discussion of approximation in value space to problems that involve stochastic uncertainty in their system equation and cost function. We will first discuss the finite horizon case, and the extension of the ideas underlying the principle of optimality and approximation in value space schemes. We will then consider the infinite horizon version of the problem, and provide an overview of the underlying theory and algorithmic methodology, in sufficient detail to allow us to speculate about infinite horizon extensions of finite horizon RL algorithms to be developed in Chapters 2-4. A more detailed discussion of the infinite horizon RL methodology will be given in Chapter 5.

† The term "Q-factor" has been used in the books [BeT96], [Ber19a], and is adopted here as well. Another term used is "action value" (at a given state). The terms "state-action value" and "Q-value" are also common in the literature. The name "Q-factor" originated in reference to the notation used in the influential Ph.D. thesis by Watkins [Wat89], which suggested the use of Q-factors in RL.

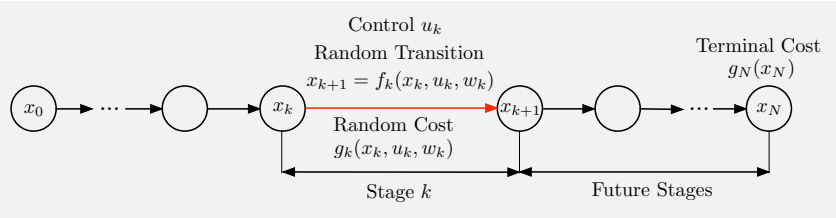


Figure 1.3.1 Illustration of an N-stage stochastic optimal control problem. Starting from state x_k, the next state under control u_k is generated randomly, according to $x_{k+1} = f_k(x_k, u_k, w_k)$, where w_k is the random disturbance, and a random stage cost $g_k(x_k, u_k, w_k)$ is incurred.

1.3.1 Finite Horizon Problems

The stochastic optimal control problem differs from the deterministic version primarily in the nature of the discrete-time dynamic system that governs the evolution of the state x_k. This system includes a random "disturbance" w_k with a probability distribution $P_k(\cdot \mid x_k, u_k)$ that may depend explicitly on x_k and u_k, but not on values of prior disturbances $w_{k-1}, \ldots, w_0$. The system has the form

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, 1, \ldots, N-1,$$

where as earlier x_k is an element of some state space and the control u_k is an element of some control space. The cost per stage is denoted by $g_k(x_k, u_k, w_k)$ and also depends on the random disturbance w_k; see Fig. 1.3.1. The control u_k is constrained to take values in a given subset $U_k(x_k)$, which depends on the current state x_k.

Given an initial state x_0 and a policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, the future states x_k and disturbances w_k are random variables with distributions defined through the system equation

$$x_{k+1} = f_k\big(x_k, \mu_k(x_k), w_k\big), \qquad k = 0, 1, \ldots, N-1,$$

and the given distributions $P_k(\cdot \mid x_k, u_k)$. Thus, for given functions g_k, $k = 0, 1, \ldots, N$, the expected cost of π starting at x_0 is

$$J_\pi(x_0) = \mathop{E}_{\substack{w_k \\ k=0,\ldots,N-1}} \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k\big(x_k, \mu_k(x_k), w_k\big) \right\},$$

where the expected value operation $E\{\cdot\}$ is taken with respect to the joint distribution of all the random variables w_k and x_k.† An optimal policy π^*

† We assume an introductory probability background on the part of the reader. For an account that is consistent with our use of probability in this book, see the text by Bertsekas and Tsitsiklis [BeT08].

is one that minimizes this cost; i.e.,

$$J_{\pi^*}(x_0) = \min_{\pi \in \Pi} J_\pi(x_0),$$

where Π is the set of all policies.

An important difference from the deterministic case is that we optimize not over control sequences $\{u_0, \ldots, u_{N-1}\}$, but rather over *policies* (also called *closed-loop control laws*, or *feedback policies*) that consist of a sequence of functions

$$\pi = \{\mu_0, \ldots, \mu_{N-1}\},$$

where μ_k maps states x_k into controls $u_k = \mu_k(x_k)$, and satisfies the control constraints, i.e., is such that $\mu_k(x_k) \in U_k(x_k)$ for all x_k. Policies are more general objects than control sequences, and in the presence of stochastic uncertainty, they can result in improved cost, since they allow choices of controls u_k that incorporate knowledge of the state x_k. Without this knowledge, the controller cannot adapt appropriately to unexpected values of the state, and as a result the cost can be adversely affected. This is a fundamental distinction between deterministic and stochastic optimal control problems.

The optimal cost depends on x_0 and is denoted by $J^*(x_0)$; i.e.,

$$J^*(x_0) = \min_{\pi \in \Pi} J_\pi(x_0).$$

We view J^* as a function that assigns to each initial state x_0 the optimal cost $J^*(x_0)$, and call it the *optimal cost function* or *optimal value function*.

Finite Horizon Stochastic Dynamic Programming

The DP algorithm for the stochastic finite horizon optimal control problem has a similar form to its deterministic version, and shares several of its major characteristics:

(a) Using tail subproblems to break down the minimization over multiple stages to single stage minimizations.

(b) Generating backwards for all k and x_k the values $J_k^*(x_k)$, which give the optimal cost-to-go starting from state x_k at stage k.

(c) Obtaining an optimal policy by minimization in the DP equations.

(d) A structure that is suitable for approximation in value space, whereby we replace J_k^* by approximations $\tilde{J}_k$, and obtain a suboptimal policy by the corresponding minimization.

DP Algorithm for Stochastic Finite Horizon Problems

Start with

$$J_N^*(x_N) = g_N(x_N),$$

and for $k = 0, \ldots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\Big\}. \tag{1.12}$$

For each x_k and k, define $\mu_k^*(x_k) = u_k^*$ where u_k^* attains the minimum in the right side of this equation. Then, the policy $\pi^* = \{\mu_0^*, \ldots, \mu_{N-1}^*\}$ is optimal.

The key fact is that starting from any initial state x_0, the optimal cost is equal to the number $J_0^*(x_0)$, obtained at the last step of the above DP algorithm. This can be proved by induction similar to the deterministic case; we will omit the proof (which incidentally involves some mathematical fine points; see the discussion of Section 1.3 in the textbook [Ber17a]).

Simultaneously with the off-line computation of the optimal cost-to-go functions $J_0^*, \ldots, J_N^*$, we can compute and store an optimal policy $\pi^* = \{\mu_0^*, \ldots, \mu_{N-1}^*\}$ by minimization in Eq. (1.12). We can then use this policy on-line to retrieve from memory and apply the control $\mu_k^*(x_k)$ once we reach state x_k. The alternative is to forego the storage of the policy π^* and to calculate the control $\mu_k^*(x_k)$ by executing the minimization (1.12) on-line.

Let us now illustrate some of the analytical and computational aspects of the stochastic DP algorithm by means of some examples. There are a few favorable cases where the optimal cost-to-go functions J_k^* and the optimal policies μ_k^* can be computed analytically. A prominent such case involves a linear system and a quadratic cost function, which is a fundamental problem in control theory. We illustrate the scalar version of this problem next. The analysis can be generalized to multidimensional systems (see optimal control textbooks such as [Ber17a]).

Example 1.3.1 (Linear Quadratic Optimal Control)

Consider a vehicle that moves on a straight-line road under the influence of a force u_k and without friction. Our objective is to maintain the vehicle's velocity at a constant level $\bar{v}$ (as in an oversimplified cruise control system). The velocity v_k at time k, after time discretization of its Newtonian dynamics and addition of stochastic noise, evolves according to

$$v_{k+1} = v_k + bu_k + w_k, \tag{1.13}$$

where w_k is a stochastic disturbance with zero mean and given variance σ^2. By introducing $x_k = v_k - \bar{v}$, the deviation between the vehicle's velocity v_k

at time k from the desired level $\bar{v}$, we obtain the system equation

$$x_{k+1} = x_k + bu_k + w_k.$$

Here the coefficient b relates to a number of problem characteristics including the weight of the vehicle and the road conditions. Moreover in practice there is friction, which introduces a coefficient $a < 1$ multiplying v_k in Eq. (1.13). For our present purposes, we assume that $a = 1$, and b is constant and known to the controller (problems involving a system with unknown and/or time-varying parameters will be discussed later). To express our desire to keep x_k near zero with relatively little force, we introduce a quadratic cost over N stages:

$$x_N^2 + \sum_{k=0}^{N-1} (x_k^2 + ru_k^2),$$

where r is a known nonnegative weighting parameter. We assume no constraints on x_k and u_k (in reality such problems include constraints, but it is common to neglect the constraints initially, and check whether they are seriously violated later).

We will apply the DP algorithm, and derive the optimal cost-to-go functions J_k^* and optimal policy. We have

$$J_N^*(x_N) = x_N^2,$$

and by applying Eq. (1.12), we obtain

$$\begin{aligned} J_{N-1}^*(x_{N-1}) &= \min_{u_{N-1}} E\big\{x_{N-1}^2 + ru_{N-1}^2 + J_N^*(x_{N-1} + bu_{N-1} + w_{N-1})\big\} \\ &= \min_{u_{N-1}} E\big\{x_{N-1}^2 + ru_{N-1}^2 + (x_{N-1} + bu_{N-1} + w_{N-1})^2\big\} \\ &= \min_{u_{N-1}} \big[x_{N-1}^2 + ru_{N-1}^2 + (x_{N-1} + bu_{N-1})^2 \\ &\qquad + 2E\{w_{N-1}\}(x_{N-1} + bu_{N-1}) + E\{w_{N-1}^2\}\big], \end{aligned}$$

and finally, using the assumption $E\{w_{N-1}\} = 0$,

$$J_{N-1}^*(x_{N-1}) = x_{N-1}^2 + \min_{u_{N-1}} \big[ru_{N-1}^2 + (x_{N-1} + bu_{N-1})^2\big] + \sigma^2. \tag{1.14}$$

The expression minimized over u_{N-1} in the preceding equation is convex quadratic in u_{N-1}, so by setting to zero its derivative with respect to u_{N-1},

$$0 = 2ru_{N-1} + 2b(x_{N-1} + bu_{N-1}),$$

we obtain the optimal policy for the last stage:

$$\mu_{N-1}^*(x_{N-1}) = -\frac{b}{r + b^2}\, x_{N-1}.$$

Substituting this expression into Eq. (1.14), we obtain with a straightforward calculation

$$J^*_{N-1}(x_{N-1}) = P_{N-1}x^2_{N-1} + \sigma^2,$$

where

$$P_{N-1} = \frac{r}{r+b^2} + 1.$$

We can now continue the DP algorithm to obtain J^*_{N-2} from J^*_{N-1}. An important observation is that J^*_{N-1} is quadratic (plus an inconsequential constant term), so with a similar calculation we can derive μ^*_{N-2} and J^*_{N-2} in closed form, as a linear and a quadratic function of x_{N-2}, respectively. This process can be continued going backwards, and it can be verified by induction that for all k, we have the optimal policy and optimal cost-to-go function in the form

$$\mu^*_k(x_k) = L_k x_k, \qquad k = 0, 1, \ldots, N-1,$$

$$J^*_k(x_k) = P_k x^2_k + \sigma^2 \sum_{t=k}^{N-1} P_{t+1}, \qquad k = 0, 1, \ldots, N-1,$$

where

$$L_k = -\frac{bP_{k+1}}{r + b^2 P_{k+1}}, \qquad k = 0, 1, \ldots, N-1,$$

and the sequence $\{P_k\}$ is generated backwards by the equation

$$P_k = \frac{rP_{k+1}}{r + b^2 P_{k+1}} + 1, \qquad k = 0, 1, \ldots, N-1,$$

starting from the terminal condition $P_N = 1$.

The process by which we obtained an analytical solution in this example is noteworthy. A little thought while tracing the steps of the algorithm will convince the reader that what simplifies the solution is the quadratic nature of the cost and the linearity of the system equation. Indeed, it can be shown in generality that when the system is linear and the cost is quadratic, the optimal policy and cost-to-go function are given by closed-form expressions, even for multi-dimensional linear systems (see [Ber17a], Section 3.1). The optimal policy is a linear function of the state, and the optimal cost function is a quadratic in the state plus a constant.

Another remarkable feature of this example, which can also be extended to multi-dimensional systems, is that the optimal policy does not depend on the variance of w_k, and remains unaffected when w_k is replaced by its mean (which is zero in our example). This is known as *certainty equivalence*, and occurs in several types of problems involving a linear system and a quadratic cost; see [Ber17a], Sections 3.1 and 4.2. For example it holds even when w_k has nonzero mean. For other problems, certainty equivalence can be used as a basis for problem approximation, e.g., assume that certainty equivalence holds (i.e., replace stochastic quantities by some typical values, such as their expected values) and apply exact DP to the resulting deterministic optimal control problem (see the discussion in Chapter 2).

The linear quadratic type of problem illustrated in the preceding example is exceptional in that it admits an elegant analytical solution. Most practical DP problems require a computational solution, and this may involve formidable difficulties. This is illustrated by the next example (which may be skipped with no loss of continuity by readers that are familiar with exact DP computations).

Example 1.3.2 (Inventory Control - Computational Illustration of Stochastic DP)

Consider a problem of ordering a quantity of a certain item at each of N stages so as to (roughly) meet a stochastic demand, while minimizing the incurred expected cost. We denote

x_k stock available at the beginning of the kth stage,

u_k stock ordered (and immediately delivered) at the beginning of the kth stage,

w_k demand during the kth stage with given probability distribution.

We assume that there is an upper bound B on the amount of stock that can be stored, and that any excess demand $(w_k - x_k - u_k)$ is turned away and is lost. As a result, the stock evolves according to the system equation

$$x_{k+1} = \max(0, x_k + u_k - w_k),$$

with the states x_k taking values in the interval $[0, B]$. This also implies the control constraint

$$0 \leq u_k \leq B - x_k.$$

The demands $w_0, \ldots, w_{N-1}$ are assumed to be independent nonnegative random variables with given probability distributions. Moreover we assume that x_k, u_k, and w_k take nonnegative integer values, so the state space is finite for all k, and consists of the integers in the interval $[0, B]$.

The cost incurred in stage k consists of two components:

(a) The purchasing cost cu_k, where c is cost per unit ordered.

(b) A cost for holding excess inventory if stock $x_k + u_k - w_k > 0$, or a cost for unfilled demand if $x_k + u_k - w_k < 0$. We write this cost generically as $r(x_k + u_k - w_k)$; as an example, for linear excess inventory/unfilled demand costs,

$$r(x_k + u_k - w_k) = \beta^+ \max(0, x_k + u_k - w_k) - \beta^- \min(0, x_k + u_k - w_k),$$

where β^+ and β^- are some positive scalars.

See Fig. 1.3.2.

There is also a terminal cost $g_N(x_N)$ for being left with inventory x_N at the end of N stages. Thus, the total cost is

$$E\left\{ g_N(x_N) + \sum_{k=0}^{N-1} \big(cu_k + r(x_k + u_k - w_k) \big) \right\}.$$

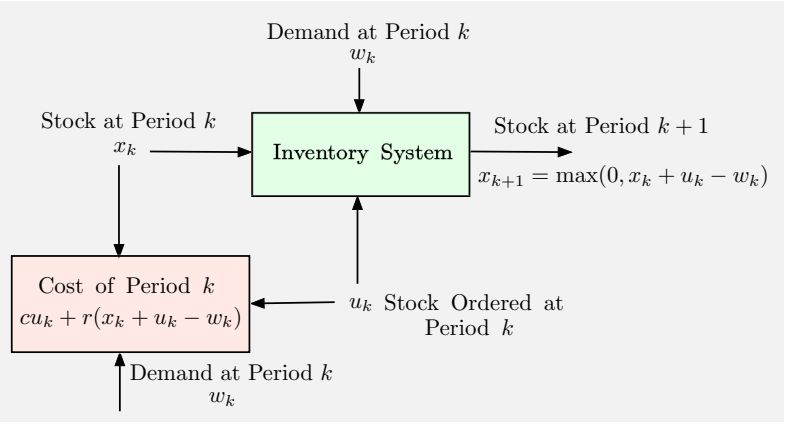


Figure 1.3.2 Inventory control example. At stage k, the current stock (state) x_k, the stock ordered (control) u_k, and the demand (random disturbance) w_k determine the cost $cu_k + r(x_k + u_k - w_k)$ of stage k and the stock $x_{k+1} = \max(0, x_k + u_k - w_k)$ at the next stage.

We want to minimize this cost by proper choice of the orders. Consistent with our closed-loop policies framework, we assume that the order u_k is chosen with knowledge of the state x_k. We are thus optimizing over ordering policies $\mu_0, \ldots, \mu_{N-1}$ that satisfy the constraint $0 \leq \mu_k(x_k) \leq B$ for all x_k and k.

The DP algorithm starts with $J_N^*(x_N) = g_N(x_N)$. At stage $N-1$, it computes the optimal cost-to-go

$$J_{N-1}^*(x_{N-1}) = \min_{0 \leq u_{N-1} \leq B - x_{N-1}} E_{w_{N-1}} \Big\{ cu_{N-1} + r(x_{N-1} + u_{N-1} - w_{N-1}) + g_N\big(\max(0, x_{N-1} + u_{N-1} - w_{N-1})\big) \Big\},$$

for all values of x_{N-1} in the interval $[0, B]$. Similarly, at stage k, the DP algorithm computes the optimal cost-to-go,

$$J_k^*(x_k) = \min_{0 \leq u_k \leq B - x_k} E_{w_k} \Big\{ cu_k + r(x_k + u_k - w_k) + J_{k+1}^*\big(\max(0, x_k + u_k - w_k)\big) \Big\}, \tag{1.15}$$

for all values of x_k in the interval $[0, B]$.

The value $J_0^*(x_0)$ is the optimal expected cost when the initial stock at time 0 is x_0. The optimal policy $\mu_k^*(x_k)$ is computed simultaneously with $J_k^*(x_k)$ from the minimization in the right-hand side of Eq. (1.15).

We will now go through the numerical calculations of the DP algorithm for a particular set of problem data. In particular, we assume that there is an upper bound of $B = 2$ units on the stock that can be stored, i.e. there is a constraint $0 \leq x_k + u_k \leq 2$. The unit purchase cost of inventory is $c = 1$, and the holding/shortage cost is

$$r(x_k + u_k - w_k) = (x_k + u_k - w_k)^2,$$

implying a penalty both for excess inventory and for unfilled demand at the end of the kth stage. Thus the cost per stage is quadratic of the form

$$g_k(x_k, u_k, w_k) = u_k + (x_k + u_k - w_k)^2.$$

The terminal cost is assumed to be 0 (unused stock at the end of the planning horizon is discarded):

$$g_N(x_N) = 0. \tag{1.16}$$

The horizon N is 3 stages, and the initial stock x_0 is 0. The demand w_k has the same probability distribution for all stages, given by

$$p(w_k = 0) = 0.1, \qquad p(w_k = 1) = 0.7, \qquad p(w_k = 2) = 0.2.$$

The state transition diagrams and the DP calculations are recorded in Fig. 1.3.3.

The starting equation for the DP algorithm is

$$J_3^*(x_3) = 0,$$

since the terminal cost is 0 [cf. Eq. (1.16)]. The algorithm takes the form [cf. Eq. (1.12)]

$$J_k^*(x_k) = \min_{\substack{0 \le u_k \le 2 - x_k \\ u_k = 0,1,2}} E_{w_k}\Big\{u_k + (x_k + u_k - w_k)^2 + J_{k+1}^*\big(\max(0, x_k + u_k - w_k)\big)\Big\},$$

where $k = 0, 1, 2$, and x_k, u_k, w_k can take the values 0, 1, and 2.

Stage 2: We compute $J_2^*(x_2)$ for each of the three possible states $x_2 = 0, 1, 2$. We have

$$\begin{aligned} J_2^*(0) &= \min_{u_2 = 0,1,2} E_{w_2}\big\{u_2 + (u_2 - w_2)^2\big\} \\ &= \min_{u_2 = 0,1,2}\big[u_2 + 0.1(u_2)^2 + 0.7(u_2 - 1)^2 + 0.2(u_2 - 2)^2\big]. \end{aligned}$$

We calculate the expected value of the right side for each of the three possible values of u_2:

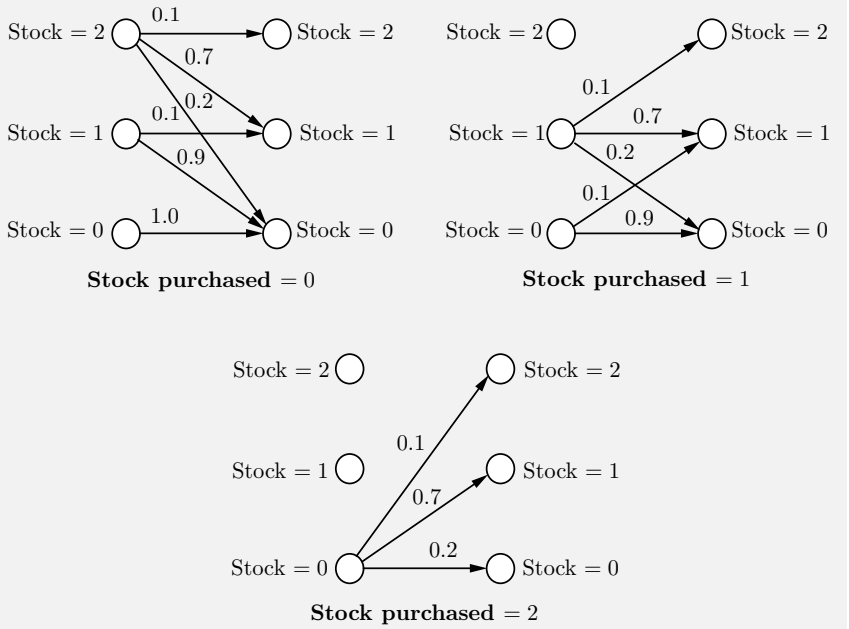
$$\begin{aligned} u_2 = 0 &: E\{\cdot\} = 0.7 \cdot 1 + 0.2 \cdot 4 = 1.5, \\ u_2 = 1 &: E\{\cdot\} = 1 + 0.1 \cdot 1 + 0.2 \cdot 1 = 1.3, \\ u_2 = 2 &: E\{\cdot\} = 2 + 0.1 \cdot 4 + 0.7 \cdot 1 = 3.1. \end{aligned}$$

Hence we have, by selecting the minimizing u_2,

$$J_2^*(0) = 1.3, \qquad \mu_2^*(0) = 1.$$

For $x_2 = 1$, we have

$$\begin{aligned} J_2^*(1) &= \min_{u_2 = 0,1} E_{w_2}\big\{u_2 + (1 + u_2 - w_2)^2\big\} \\ &= \min_{u_2 = 0,1}\big[u_2 + 0.1(1 + u_2)^2 + 0.7(u_2)^2 + 0.2(u_2 - 1)^2\big]. \end{aligned}$$



Stock	Stage 0 Cost-to-go	Stage 0 Optimal stock to purchase	Stage 1 Cost-to-go	Stage 1 Optimal stock to purchase	Stage 2 Cost-to-go	Stage 2 Optimal stock to purchase
0	3.7	1	2.5	1	1.3	1
1	2.7	0	1.5	0	0.3	0
2	2.818	0	1.68	0	1.1	0

Figure 1.3.3 The DP results for Example 1.3.2. The state transition diagrams for the different values of stock purchased (control) are shown. The numbers next to the arcs are the transition probabilities. The control $u = 1$ is not available at state 2 because of the limitation $x_k + u_k \leq 2$. Similarly, the control $u = 2$ is not available at states 1 and 2. The results of the DP algorithm are given in the table.

The expected value in the right side is

$$u_2 = 0 : E\{\cdot\} = 0.1 \cdot 1 + 0.2 \cdot 1 = 0.3,$$
$$u_2 = 1 : E\{\cdot\} = 1 + 0.1 \cdot 4 + 0.7 \cdot 1 = 2.1.$$

Hence

$$J_2^*(1) = 0.3, \qquad \mu_2^*(1) = 0.$$

For $x_2 = 2$, the only admissible control is $u_2 = 0$, so we have

$$J_2^*(2) = E_{w_2}\big\{(2 - w_2)^2\big\} = 0.1 \cdot 4 + 0.7 \cdot 1 = 1.1,$$

$$J_2^*(2) = 1.1, \qquad \mu_2^*(2) = 0.$$

Stage 1: Again we compute $J_1^*(x_1)$ for each of the three possible states $x_1 = 0, 1, 2$, using the values $J_2^*(0)$, $J_2^*(1)$, $J_2^*(2)$ obtained in the previous stage. For $x_1 = 0$, we have

$$J_1^*(0) = \min_{u_1=0,1,2} E_{w_1}\Big\{u_1 + (u_1 - w_1)^2 + J_2^*\big(\max(0, u_1 - w_1)\big)\Big\},$$

$$u_1 = 0 : E\{\cdot\} = 0.1 \cdot J_2^*(0) + 0.7\big(1 + J_2^*(0)\big) + 0.2\big(4 + J_2^*(0)\big) = 2.8,$$
$$u_1 = 1 : E\{\cdot\} = 1 + 0.1\big(1 + J_2^*(1)\big) + 0.7 \cdot J_2^*(0) + 0.2\big(1 + J_2^*(0)\big) = 2.5,$$
$$u_1 = 2 : E\{\cdot\} = 2 + 0.1\big(4 + J_2^*(2)\big) + 0.7\big(1 + J_2^*(1)\big) + 0.2 \cdot J_2^*(0) = 3.68,$$

$$J_1^*(0) = 2.5, \qquad \mu_1^*(0) = 1.$$

For $x_1 = 1$, we have

$$J_1^*(1) = \min_{u_1=0,1} E_{w_1}\Big\{u_1 + (1 + u_1 - w_1)^2 + J_2^*\big(\max(0, 1 + u_1 - w_1)\big)\Big\},$$

$$u_1 = 0 : E\{\cdot\} = 0.1\big(1 + J_2^*(1)\big) + 0.7 \cdot J_2^*(0) + 0.2\big(1 + J_2^*(0)\big) = 1.5,$$
$$u_1 = 1 : E\{\cdot\} = 1 + 0.1\big(4 + J_2^*(2)\big) + 0.7\big(1 + J_2^*(1)\big) + 0.2 \cdot J_2^*(0) = 2.68,$$

$$J_1^*(1) = 1.5, \qquad \mu_1^*(1) = 0.$$

For $x_1 = 2$, the only admissible control is $u_1 = 0$, so we have

$$\begin{aligned} J_1^*(2) &= E_{w_1}\Big\{(2 - w_1)^2 + J_2^*\big(\max(0, 2 - w_1)\big)\Big\} \\ &= 0.1\big(4 + J_2^*(2)\big) + 0.7\big(1 + J_2^*(1)\big) + 0.2 \cdot J_2^*(0) \\ &= 1.68, \end{aligned}$$

$$J_1^*(2) = 1.68, \qquad \mu_1^*(2) = 0.$$

Stage 0: Here we need to compute only $J_0^*(0)$ since the initial state is known to be 0. We have

$$J_0^*(0) = \min_{u_0=0,1,2} E_{w_0}\Big\{u_0 + (u_0 - w_0)^2 + J_1^*\big(\max(0, u_0 - w_0)\big)\Big\},$$

$$u_0 = 0 : E\{\cdot\} = 0.1 \cdot J_1^*(0) + 0.7\big(1 + J_1^*(0)\big) + 0.2\big(4 + J_1^*(0)\big) = 4.0,$$
$$u_0 = 1 : E\{\cdot\} = 1 + 0.1\big(1 + J_1^*(1)\big) + 0.7 \cdot J_1^*(0) + 0.2\big(1 + J_1^*(0)\big) = 3.7,$$
$$u_0 = 2 : E\{\cdot\} = 2 + 0.1\big(4 + J_1^*(2)\big) + 0.7\big(1 + J_1^*(1)\big) + 0.2 \cdot J_1^*(0) = 4.818,$$

$$J_0^*(0) = 3.7, \qquad \mu_0^*(0) = 1.$$

If the initial state were not known a priori, we would have to compute in a similar manner $J_0^*(1)$ and $J_0^*(2)$, as well as the minimizing u_0. The reader may verify that these calculations yield

$$J_0^*(1) = 2.7, \qquad \mu_0^*(1) = 0,$$

$$J_0^*(2) = 2.818, \qquad \mu_0^*(2) = 0.$$

Thus the optimal ordering policy for each stage is to order one unit if the current stock is zero and order nothing otherwise. The results of the DP algorithm are given in lookup table form in Fig. 1.3.3.

1.3.2 Approximation in Value Space for Stochastic DP

Generally the computation of the optimal cost-to-go functions J_k^* can be very time-consuming or impossible. One of the principal RL methods to deal with this difficulty is approximation in value space. Here approximations $\tilde{J}_k$ are used in place of J_k^*, similar to the deterministic case; cf. Eqs. (1.7) and (1.9).

Approximation in Value Space - Use of $\tilde{J}_k$ in Place of J_k^*

At any state x_k encountered at stage k, set

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} E_{w_k} \Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big\}. \tag{1.17}$$

The motivation for approximation in value space for stochastic DP problems is similar to the one for deterministic problems. The one-step lookahead minimization (1.17) needs to be performed only for the N states $x_0, \ldots, x_{N-1}$ that are encountered during the on-line control of the system, and not for every state within the potentially enormous state space.

Our discussion of rollout (cf. Section 1.2.3) also applies to stochastic problems: we select $\tilde{J}_k$ to be the cost function of a suitable base policy (perhaps with some approximation). Note that any policy can be used on-line as base policy, including policies obtained by a sophisticated off-line procedure, using for example neural networks and training data.† The

† The principal role of neural networks within the context of this book is to provide the means for approximating various target functions from input-output data. This includes cost functions and Q-factors of given policies, and optimal cost-to-go functions and Q-factors; in this case the neural network is referred to as a *value network* (sometimes the alternative term *critic network* is also used). In

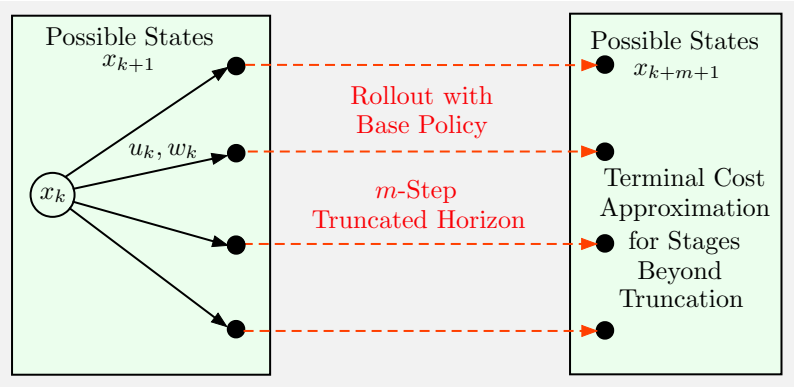


Figure 1.3.4 Schematic illustration of truncated rollout. One-step lookahead is followed by simulation of the base policy for m steps, and an approximate cost $\tilde{J}_{k+m+1}(x_{k+m+1})$ is added to the cost of the simulation, which depends on the state x_{k+m+1} obtained at the end of the rollout. If the base policy simulation is omitted (i.e., $m = 0$), one recovers the general approximation in value space scheme (1.17). There are also multistep lookahead versions of truncated rollout (see Chapter 2).

rollout algorithm has the cost improvement property, whereby it yields an improved cost relative to its underlying base policy.

A major variant of rollout is *truncated rollout*, which combines the use of one-step optimization, simulation of the base policy for a certain number of steps m, and then adds an approximate cost $\tilde{J}_{k+m+1}(x_{k+m+1})$ to the cost of the simulation, which depends on the state x_{k+m+1} obtained at the end of the rollout (see Chapter 2). Note that if one foregoes the use of a base policy (i.e., $m = 0$), one recovers as a special case the general approximation in value space scheme (1.17); see Fig. 1.3.4. Note also that versions of truncated rollout with multistep lookahead minimization are possible. They will be discussed in subsequent chapters. The terminal cost approximation is necessary in infinite horizon problems, since an infinite number of stages of the base policy rollout is impossible. However, even for finite horizon problems it may be necessary and/or beneficial to artificially truncate the rollout horizon. This is particularly true if the truncated rollout is implemented with the aid of Monte Carlo simulation, in which case simulation noise may become a concern.

Another helpful point of view is to interpret m-step truncated rollout as an approximate m-step extension of the length of the multistep

---

other cases the neural network represents a policy viewed as a function from state to control, in which case it is called a *policy network* (the alternative term *actor network* is also used). The training methods for constructing the cost function, Q-factor, and policy approximations themselves from data are mostly based on optimization and regression, and will be discussed in Chapter 4.

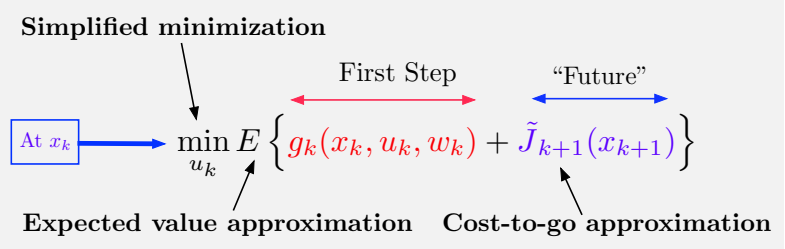


Figure 1.3.5 Schematic illustration of approximation in value space for stochastic problems, and the three approximations involved in its design. Typically these approximations can be designed independently of each other. There are also multistep lookahead versions of approximation in value space (see Chapter 2).

minimization. Since a long lookahead minimization is likely beneficial, it follows that unless the terminal cost approximation is very good (i.e., $\tilde{J}_{k+m+1} \approx J^*_{k+m+1}$), a large combined number of multistep lookahead minimization and rollout steps is also likely to be beneficial.

Note that we may be useful to simplify the lookahead minimization over $u_k \in U_k(x_k)$ [cf. Eq. (1.12)]. In particular, in the multiagent case where the control consists of multiple components, $u_k = (u_k^1, \ldots, u_k^m)$, a sequence of m single component minimizations can be used instead, with potentially enormous computational savings resulting.

There is one additional issue in approximation in value space for stochastic problems: the computation of the expected value in Eq. (1.17) may be very time-consuming. Then one may consider approximations in the computation of this expected value, based for example on Monte Carlo simulation or other schemes. Some of the possibilities along this line will be discussed in the next chapter.

Figure 1.3.5 illustrates the three approximations involved in approximation in value space for stochastic problems: *cost-to-go approximation, expected value approximation, and simplified minimization*. They may be designed largely independently of each other, and with a variety of methods. Most of the discussion in this book will revolve around different ways to organize these three approximations, in the context of finite and infinite horizon problems, and some of their special cases.

Q-Factors for Stochastic Problems

We can define optimal Q-factors for a stochastic problem, similar to the case of deterministic problems [cf. Eq. (1.10)], as the expressions that are minimized in the right-hand side of the stochastic DP equation (1.12). They are given by

$$Q_k^*(x_k,u_k) = E_{w_k}\Big\{g_k(x_k,u_k,w_k) + J_{k+1}^*\big(f_k(x_k,u_k,w_k)\big)\Big\}. \qquad (1.18)$$

The optimal cost-to-go functions J_k^* can be recovered from the optimal Q-factors Q_k^* by means of

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} Q_k^*(x_k, u_k),$$

and the DP algorithm can be written in terms of Q-factors as

$$Q_k^*(x_k, u_k) = E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + \min_{u_{k+1} \in U_{k+1}(f_k(x_k, u_k, w_k))} Q_{k+1}^*\big(f_k(x_k, u_k, w_k), u_{k+1}\big)\Big\}.$$

Similar to the deterministic case, Q-learning involves the calculation of either the optimal Q-factors (1.18) or approximations $\tilde{Q}_k(x_k, u_k)$. The approximate Q-factors may be obtained using approximation in value space schemes, and can be used to obtain approximately optimal policies through the Q-factor minimization

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k). \tag{1.19}$$

In Chapter 4, we will discuss the use of neural networks in such approximations.

Cost Versus Q-Factor Approximations - Robustness and On-Line Replanning

We have seen that it is possible to implement approximation in value space by using cost function approximations [cf. Eq. (1.17)] or by using Q-factor approximations [cf. Eq. (1.19)], so the question arises which one to use in a given practical situation. One important consideration is the facility of obtaining suitable cost or Q-factor approximations. This depends largely on the problem and also on the availability of data on which the approximations can be based. However, there are some other major considerations.

In particular, the cost function approximation scheme

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big)\Big\},$$

has an important disadvantage: *the expected value above needs to be computed on-line for all $u_k \in U_k(x_k)$, and this may involve substantial computation*. On the other hand it also has an important advantage in situations where the system function f_k, the cost per stage g_k, or the control constraint set $U_k(x_k)$ can change as the system is operating. Assuming that the new f_k, g_k, or $U_k(x_k)$ become known to the controller at time k, *on-line replanning may be used, and this may improve substantially the robustness*

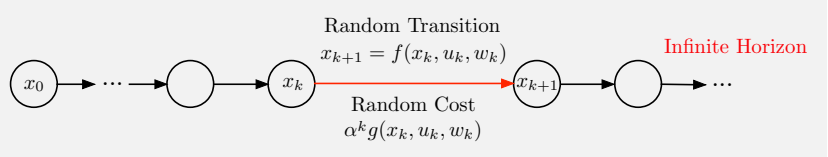


Figure 1.3.6 Illustration of an infinite horizon problem. The system and cost per stage are stationary, except for the use of a discount factor α. If $\alpha = 1$, there is typically a special cost-free termination state that we aim to reach.

of the approximation in value space scheme, as discussed earlier for deterministic problems.

By comparison, the Q-factor function approximation scheme (1.19) does not allow for on-line replanning. On the other hand, for problems where there is no need for on-line replanning, Q-factor approximation does not require the on-line evaluation of expected values and may allow fast on-line computation of the minimizing control $\tilde{\mu}_k(x_k)$ via Eq. (1.19).

1.3.3 Infinite Horizon Problems - An Overview

We will now provide an outline of infinite horizon stochastic DP with an emphasis on its aspects that relate to our RL/approximation methods. In Chapter 5 we will deal with two types of infinite horizon stochastic problems, where we aim to minimize the total cost over an infinite number of stages, given by

$$J_\pi(x_0) = \lim_{N\to\infty} \mathop{E}_{\substack{w_k \\ k=0,1,\ldots}} \left\{ \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k), w_k\big) \right\}; \tag{1.20}$$

see Fig. 1.3.6. Here, $J_\pi(x_0)$ denotes the cost associated with an initial state x_0 and a policy $\pi = \{\mu_0, \mu_1, \ldots\}$, and α is a scalar in the interval $(0, 1]$.

When α is strictly less that 1, it has the meaning of a *discount factor*, and its effect is that future costs matter to us less than the same costs incurred at the present time. Among others, a discount factor guarantees that the limit defining $J_\pi(x_0)$ exists and is finite (assuming that the range of values of the stage cost g is bounded). This is a nice mathematical property that makes discounted problems analytically and algorithmically tractable.

Thus, by definition, the infinite horizon cost of a policy is the limit of its finite horizon costs as the horizon tends to infinity. The two types of problems that we will focus on in Chapter 5 are:

(a) *Stochastic shortest path problems* (SSP for short). Here, $\alpha = 1$ but there is a special cost-free termination state; once the system reaches that state it remains there at no further cost. In some types of problems, the termination state may represent a goal state that we are

trying to reach at minimum cost, while in others it may be a state that we are trying to avoid for as long as possible. We will mostly assume a problem structure such that termination is inevitable under all policies. Thus the horizon is in effect finite, but its length is random and may be affected by the policy being used. A significantly more complicated type of SSP problems arises when termination can be guaranteed only for a subset of policies, which includes all optimal policies. Some common types of SSP problems belong to this category, including deterministic shortest path problems that involve graphs with cycles.

(b) *Discounted problems*. Here, $\alpha < 1$ and there need not be a termination state. However, we will see that a discounted problem with a finite number of states can be readily converted to an SSP problem. This can be done by introducing an artificial termination state to which the system moves with probability $1 - \alpha$ at every state and stage, thus making termination inevitable. As a result, algorithms and analysis for SSP problems can be easily adapted to discounted problems. Moreover, a common line of analysis and algorithmic development may often be used for both SSP and discounted problems, as we will see in Chapter 5.

In our theoretical and algorithmic analysis of Chapter 5, we will focus on finite state problems. Much of our infinite horizon methodology applies in principle to problems with continuous spaces problems as well, but there are significant exceptions and mathematical complications (the abstract DP monograph [Ber18a] takes a closer look at such problems). Still we will use the infinite horizon methodology in continuous spaces settings with somewhat limited justification, as an extension of the finite-state methods that we will justify more rigorously.

Infinite Horizon Theory - Value Iteration

There are several analytical and computational issues regarding our infinite horizon problems. Many of them revolve around the relation between the optimal cost function J^* of the infinite horizon problem and the optimal cost functions of the corresponding N-stage problems.

In particular, consider the undiscounted case ($\alpha = 1$) and let $J_N(x)$ denote the optimal cost of the problem involving N stages, initial state x, cost per stage $g(x, u, w)$, and zero terminal cost. This cost is generated after N iterations of the DP algorithm

$$J_{k+1}(x) = \min_{u \in U(x)} E_w \Big\{ g(x, u, w) + J_k\big(f(x, u, w)\big) \Big\}, \qquad k = 0, 1, \ldots, \tag{1.21}$$

starting from $J_0(x) \equiv 0$. The algorithm (1.21) is just the DP algorithm with the time indexing reversed, and is known as the *value iteration* algorithm

(VI for short). Since the infinite horizon cost of a given policy is, by definition, the limit of the corresponding N-stage costs as $N \to \infty$, it is natural to speculate that:

(1) The optimal infinite horizon cost is the limit of the corresponding N-stage optimal costs as $N \to \infty$; i.e.,

$$J^*(x) = \lim_{N\to\infty} J_N(x) \tag{1.22}$$

for all states x.

(2) The following equation should hold for all states x,

$$J^*(x) = \min_{u\in U(x)} E_w\Big\{g(x,u,w) + J^*\big(f(x,u,w)\big)\Big\}. \tag{1.23}$$

This is obtained by taking the limit as $N \to \infty$ in the VI algorithm (1.21) using Eq. (1.22). The preceding equation, called *Bellman's equation*, is really a system of equations (one equation per state x), which has as solution the optimal costs-to-go of all the states.

(3) If $\mu(x)$ attains the minimum in the right-hand side of the Bellman equation (1.23) for each x, then the policy $\{\mu, \mu, \ldots\}$ should be optimal. This type of policy is called *stationary*. Intuitively, optimal policies can be found within this class of policies, since optimization of the future costs (the tail subproblem) when starting at a given state looks the same regardless of the time when we start.

All three of the preceding results hold for finite-state SSP problems under reasonable assumptions. They also hold for discounted problems in suitably modified form that incorporates the discount factor, provided the cost per stage function g is bounded over the set of possible values of (x, u, w) (see [Ber12], Chapter 1). In particular, the discounted version of the VI algorithm of Eq. (1.21) is†

$$J_{k+1}(x) = \min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J_k\big(f(x,u,w)\big)\Big\}, \qquad k = 0, 1, \ldots, \tag{1.24}$$

† This is again the finite horizon DP algorithm. In particular, consider the N-stages problem and let $V_{N-k}(x)$ be the optimal cost-to-go starting at x with k stages to go, and with terminal cost equal to 0. Applying DP, we have

$$V_{N-k}(x) = \min_{u\in U(x)} E_w\Big\{\alpha^{N-k} g(x,u,w) + V_{N-k+1}\big(f(x,u,w)\big)\Big\}, \quad V_N(x) = 0,$$

for all x. By dividing by α^{N-k} and defining $J_k(x) = V_{N-k+1}(x)/\alpha^{N-k+1}$, we obtain the discounted version of the VI algorithm.

while the discounted version of Bellman's equation [cf. Eq. (1.23)] is

$$J^*(x) = \min_{u \in U(x)} E_w \Big\{ g(x,u,w) + \alpha J^*\big(f(x,u,w)\big) \Big\}. \tag{1.25}$$

The VI algorithm of Eq. (1.24) simply expresses the fact that the optimal cost for $k+1$ stages is obtained by minimizing the sum of the first stage cost and the optimal cost for the next k stages starting from the next state $f(x,u,w)$. The latter cost, however, should be discounted by α in view of the cost definition (1.20). The intuitive interpretation of the Bellman equation (1.25) is that it is the limit as $k \to \infty$ of the VI algorithm (1.24) assuming that $J_k \to J^*$.

The VI algorithm is also often valid, in the sense that $J_k \to J^*$, even if the initial function J_0 is nonzero. The motivation for a different choice of J_0 is faster convergence to J^*; generally the convergence is faster as J_0 is chosen closer to J^*.

We will now provide two examples that illustrate how the infinite horizon DP theory can be used for an analytical solution of the problem. The first example involves continuous state and control spaces and is based on the linear quadratic formulation (cf. Example 1.3.1). The second example involves discrete state and control spaces and is of the SSP type. These examples involve a favorable special structure, which allows an analytical solution. Such problems are rare. Most practical DP problems require a computational solution.

Example 1.3.3 (Linear Quadratic Optimal Control)

Consider the infinite horizon version of the linear quadratic problem of Example 1.3.1, assuming a discount factor $\alpha < 1$. Thus the cost function is given by

$$\lim_{N \to \infty} \underset{\substack{w_k \\ k=0,1,\ldots}}{E} \left\{ \sum_{k=0}^{N-1} \alpha^k (x_k^2 + r u_k^2) \right\},$$

and the VI algorithm (1.24) has the form

$$J_{k+1}(x) = \min_u E_w \big\{ x^2 + r u^2 + \alpha J_k(x + bu + w) \big\}, \quad k = 0, 1, \ldots. \tag{1.26}$$

Since the VI iterates are cost functions of linear quadratic finite horizon problems, it follows from the analysis of the finite horizon case of Example 1.3.1 that J_k is a convex quadratic in the state plus a constant. A simple modification of the calculations of that example (to incorporate the effect of the discount factor) yields the VI iterates as

$$J_{k+1}(x) = K_k x^2 + \sigma^2 \sum_{t=0}^{k-1} \alpha^{k-t} K_t, \qquad k = 0, 1, \ldots, \tag{1.27}$$

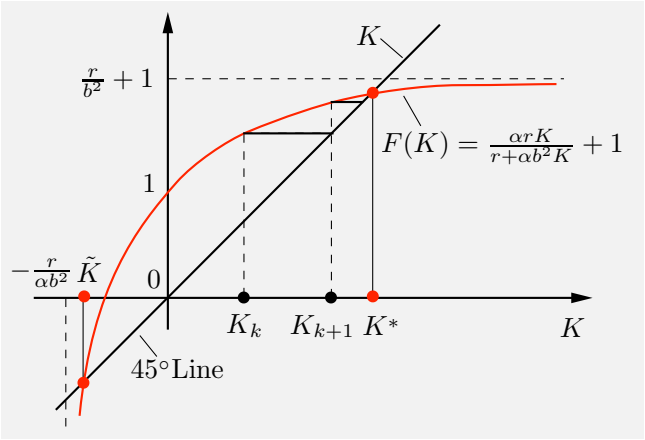


Figure 1.3.7 Graphical construction of the two solutions of Bellman's equation, and proof of the convergence of the VI algorithm (1.27)-(1.28) for the linear quadratic Example 1.3.3, starting from a quadratic initial function. The sequence $\{K_k\}$ generated by the VI algorithm is given by $K_{k+1} = F(K_k)$ where F is the function given by

$$F(K) = \frac{\alpha rK}{r + \alpha b^2 K} + 1.$$

Because F is concave and monotonically increasing in the interval $(-r/\alpha b^2, \infty)$ and "flattens out" as $K \to \infty$, as shown in the figure, the equation $K = F(K)$ has one positive solution K^* and one negative solution $\tilde{K}$. The iteration $K_{k+1} = F(K_k)$ [cf. Eq. (1.28)] converges to K^* starting from anywhere in the interval $(\tilde{K}, \infty)$ as shown in the figure. This iteration is essentially equivalent to the VI algorithm (1.26) with a quadratic starting function.

starting from the initial condition $J_0(x) = 0$, where the sequence $\{K_k\}$ is generated by the equation

$$K_{k+1} = \frac{\alpha rK_k}{r + \alpha b^2 K_k} + 1, \qquad k = 0, 1, \ldots, \tag{1.28}$$

starting from the initial condition $K_0 = 1$ (K_k corresponds to P_{N-k} in Example 1.3.1 because of the reversal of time indexing).

It can be verified that the sequence $\{K_k\}$ is convergent to the scalar K^*, which is the unique positive solution of the equation

$$K = \frac{\alpha rK}{r + \alpha b^2 K} + 1, \tag{1.29}$$

the limit as $k \to \infty$ of Eq. (1.28) (a graphical explanation is given in Fig. 1.3.7). Equation (1.29) is a one-dimensional special case of the famous discrete-time Riccati equation from control theory. Moreover, the VI algorithm (1.27) converges to the optimal cost function, which is the quadratic function

$$J^*(x) = K^* x^2 + \frac{\alpha}{1-\alpha} K^* \sigma^2; \tag{1.30}$$

cf. Eq. (1.27). It can be verified by straightforward calculation that J^* satisfies Bellman's equation,

$$J^*(x) = \min_u E_w\big\{x^2 + ru^2 + \alpha J^*(x + bu + w)\big\}.$$

The optimal policy is obtained by the preceding minimization:

$$\mu^*(x) \in \arg\min_u E_w\big\{x^2 + ru^2 + \alpha K^*(x + bu + w)^2\big\} + \frac{\alpha^2}{1-\alpha} K^* \sigma^2.$$

By setting the derivative of the minimized expression to 0, it can be seen that μ^* is linear and stationary of the form

$$\mu^*(x) = Lx, \tag{1.31}$$

where

$$L = -\frac{\alpha b K^*}{r + \alpha b^2 K^*}. \tag{1.32}$$

This is the limit of the nonstationary optimal policy derived in Example 1.3.1 for the finite horizon version of the problem.

The preceding derivation can be extended to general multidimensional linear quadratic infinite horizon discounted problems, and can be established by a rigorous analysis (see [Ber12], Chapter 4). The linearity and simplicity of the optimal policy (1.31) is noteworthy, and a major reason for the popularity of the linear quadratic formulation.

The undiscounted version ($\alpha = 1$) of the infinite horizon linear quadratic problem is also of interest. A mathematical complication here is that the optimal cost function (1.30) becomes infinite as $\alpha \to 1$, when $\sigma^2 > 0$. On the other hand, for a deterministic problem ($w_k \equiv 0$ or equivalently $\sigma^2 = 0$) the informal derivation given above goes through, and the optimal controller is still linear of the form (1.31), where now L is the limit as $\alpha \to 1$ of the discounted version (1.32):

$$L = -\frac{bK^*}{r + b^2 K^*}, \tag{1.33}$$

where K^* is the positive solution of the Riccati equation

$$K = \frac{rK}{r + b^2 K} + 1.$$

(This quadratic equation in K has two solutions, one positive and one negative, when $r > 0$; see Fig. 1.3.7.)

The linear policy $\mu^*(x) = Lx$, with the coefficient L given by Eq. (1.33), is also optimal for a stochastic linear quadratic optimal control problem with the average cost criterion, which is not discussed in this book, but is covered elsewhere in the literature (see e.g., [Ber12], Section 5.6.5, and the references quoted there).

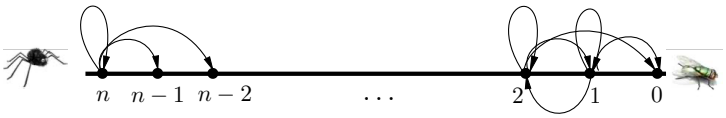


Figure 1.3.8 Illustration of transition probabilities of the spider and fly problem. The state is the distance between spider and fly, and cannot increase, except when the spider and fly are one unit apart. There are $n + 1$ states, with state 0 being the termination state.

Example 1.3.4 (Spider and Fly)

A spider and a fly move along a straight line at times $k = 0, 1, \ldots$ The initial positions of the fly and the spider are integer. At each stage, the fly moves one unit to the left with probability p, one unit to the right with probability p, and stays where it is with probability $1-2p$. The spider, knows the position of the fly at the beginning of each stage, and will always move one unit towards the fly if its distance from the fly is more that one unit. If the spider is one unit away from the fly, it will either move one unit towards the fly or stay where it is. If the spider and the fly land in the same position at the end of a stage, then the spider captures the fly and the process terminates. The spider's objective is to capture the fly in minimum expected time; see Fig. 1.3.8.

We view as state the distance between spider and fly. Then the problem can be formulated as an SSP problem with states $0, 1, \ldots, n$, where n is the initial distance. State 0 is a termination state where the spider captures the fly. Let us denote $p_{1y}(m)$ and $p_{1y}(\overline{m})$ the transition probabilities from state 1 to state y under the two spider choices/controls of moving and not moving, respectively, and let us denote by p_{xy} the transition probabilities from a state $x \geq 2$.† We have

$$p_{xx} = p, \quad p_{x(x-1)} = 1 - 2p, \quad p_{x(x-2)} = p, \qquad x \geq 2,$$

$$p_{11}(m) = 2p, \quad p_{10}(m) = 1 - 2p,$$

$$p_{12}(\overline{m}) = p, \quad p_{11}(\overline{m}) = 1 - 2p, \quad p_{10}(\overline{m}) = p,$$

with all other transition probabilities being 0.

For states $x \geq 2$, Bellman's equation is written as

$$J^*(x) = 1 + pJ^*(x) + (1 - 2p)J^*(x - 1) + pJ^*(x - 2), \quad x \geq 2, \tag{1.34}$$

† Transition probabilities are a common way to describe stochastic finite-state systems, and will be used in Chapter 5. In particular, given the transition probabilities $p_{xy}(u)$, i.e., the conditional probabilities of moving from state x to state y under control u, an equivalent system equation description is $x_{k+1} = w_k$, where the disturbance w_k is generated according to the conditional distribution

$$p(w_k = y \mid x_k = x,\, u_k = u) = p_{xy}(u),$$

for all possible values of (x, y, u).

where for the termination state 0, we have $J^*(0) = 0$. The only state where the spider has a choice is when it is one unit away from the fly, and for that state Bellman's equation is given by

$$J^*(1) = 1 + \min\Big[2pJ^*(1),\, pJ^*(2) + (1-2p)J^*(1)\Big], \tag{1.35}$$

where the first and the second expression within the bracket above are associated with the spider moving and not moving, respectively. By writing Eq. (1.34) for $x = 2$, we obtain

$$J^*(2) = 1 + pJ^*(2) + (1-2p)J^*(1),$$

from which

$$J^*(2) = \frac{1}{1-p} + \frac{(1-2p)J^*(1)}{1-p}.$$

Substituting this expression in Eq. (1.35), we obtain

$$J^*(1) = 1 + \min\left[2pJ^*(1),\, \frac{p}{1-p} + \frac{p(1-2p)J^*(1)}{1-p} + (1-2p)J^*(1)\right],$$

or equivalently,

$$J^*(1) = 1 + \min\left[2pJ^*(1),\, \frac{p}{1-p} + \frac{(1-2p)J^*(1)}{1-p}\right]. \tag{1.36}$$

Thus Bellman's equation at state 1 can be reduced to the one-dimensional Eq. (1.36), and once this equation is solved it can be used to solve the Bellman Eq. (1.34) for all $x > 1$. It is also possible to derive analytically the optimal policy, but the derivation is somewhat lengthy, so we refer to [Ber17a], Section 5.2, for the details. It turns out that the optimal policy is simple and intuitive: for $p \leq 1/3$ (when the fly does not move a lot), it is optimal for the spider to move towards the fly, and it is optimal to stay motionless otherwise.

Despite its simplicity, the spider-and-fly problem of the preceding example can become quite difficult even with seemingly minor modifications. For example if the fly is "faster" than the spider, and can move up to two units to the left or right with positive probabilities, the distance between spider and fly may become arbitrarily large with positive probability, and the number of states becomes infinite. Then, not only the Bellman equation is hard to solve analytically, but also the standard theory of SSP problems does not apply (a more complex extension of the theory that applies to infinite state space problems is needed; see [Ber18a], Chapter 4).

As another example, suppose that the spider has "blurry" vision and cannot see the exact location of the fly. Then we may formulate the problem as one where the choice of the fly depends on all of its past observations. We will consider later such problems, called POMDP for (partially observed Markovian decision problem), and we will see that they are far more difficult, both analytically and computationally, than the problems we have discussed so far.

1.3.4 Infinite Horizon - Approximation in Value Space

A principal RL approach to deal with the often intractable exact computation of J^* is approximation in value space. Here an approximation $\tilde{J}$ is used in place of J^*, similar to the finite horizon case.

Approximation in Value Space - Use of $\tilde{J}$ in Place of J^*

For any state x, use a control $\tilde{\mu}(x)$ obtained by the one-step lookahead minimization

$$\tilde{\mu}(x) \in \arg \min_{u_k \in U(x)} E_w \Big\{ g(x,u,w) + \tilde{J}\big(f(x,u,w)\big) \Big\}. \tag{1.37}$$

Some insight into Bellman's equation, the VI algorithm, approximation in value space, and some of the properties of the corresponding one-step lookahead policy $\tilde{\mu}$, can be obtained with the help of geometric constructions, given in Figs. 1.3.9 and 1.3.10. To understand these figures we need some abstract notation (we will use extensively this notation in Chapter 5). In particular, we denote by TJ the function in the right-hand side of Bellman's equation. Its value at state x is given by

$$(TJ)(x) = \min_{u \in U(x)} E_w \Big\{ g(x,u,w) + \alpha J\big(f(x,u,w)\big) \Big\}. \tag{1.38}$$

Also for each policy μ, we introduce the corresponding function $T_\mu J$, which has value at x given by

$$(T_\mu J)(x) = E_w \Big\{ g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big) \Big\}. \tag{1.39}$$

Thus T and T_μ can be viewed as operators (referred to as the *Bellman operators*), which map cost functions J to other cost functions (TJ or $T_\mu J$, respectively).

An important property of T and T_μ is that they are *monotone*, in the sense that if J and J' are two functions of x such that

$$J(x) \geq J'(x), \qquad \text{for all } x,$$

then we have

$$(TJ)(x) \geq (TJ')(x), \quad (T_\mu J)(x) \geq (T_\mu J')(x), \qquad \text{for all } x.$$

Note that the Bellman equation $J = T_\mu J$ is linear, while the function TJ can be written as $\min_\mu T_\mu J$ and its components are concave. For example, assume two states 1 and 2, and two controls u and v. Consider the

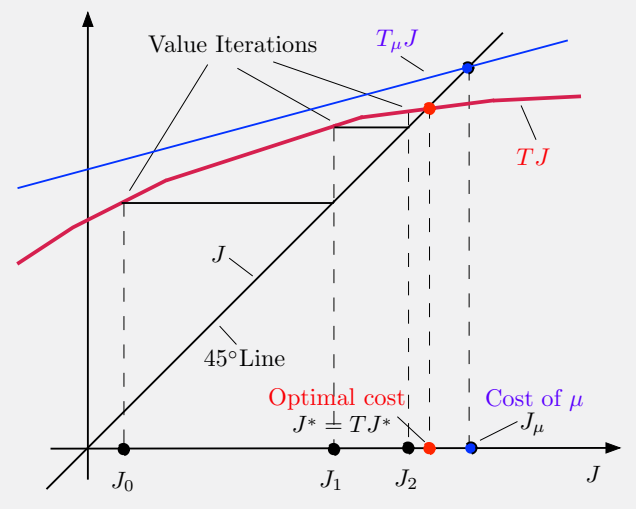


Figure 1.3.9 Geometric interpretation of the Bellman operators T_μ and T, the Bellman equations, and the VI algorithm. The function $T_\mu J$ for each policy μ is linear, while the function TJ can be written as $\min_\mu T_\mu J$ and its components are concave. The functions J, J^*, TJ, etc, are multidimensional (they have as many scalar components as there are states), but they are shown projected onto one dimension. The optimal cost J^* satisfies $J^* = TJ^*$, so it is obtained from the intersection of the graph of TJ and the 45 degree line shown. The VI sequence $\{J_k\}$, generated by $J_{k+1} = TJ_k$, is obtained with the staircase construction shown, and asymptotically converges to J^* (cf. the linear-quadratic case of Fig. 1.3.7).

policy μ that applies control u at state 1 and control v at state 2. Then, using transition probability notation, the operator T_μ takes the form

$$(T_\mu J)(1) = \sum_{j=1}^{2} p_{1j}(u)\big(g(1,u,j) + \alpha J(j)\big),$$

$$(T_\mu J)(2) = \sum_{j=1}^{2} p_{2j}(v)\big(g(2,v,j) + \alpha J(j)\big),$$

where $p_{ij}(u)$ and $p_{ij}(v)$ are the probabilities that the next state will be j, when the current state is i, and the control is u or v, respectively. Clearly, $(T_\mu J)(1)$ and $(T_\mu J)(2)$ are linear functions of J. Also the operator T of the Bellman equation $J = TJ$ takes the form

$$(TJ)(1) = \min \left[\sum_{j=1}^{2} p_{1j}(u)\big(g(1,u,j) + \alpha J(j)\big), \sum_{j=1}^{2} p_{1j}(v)\big(g(1,v,j) + \alpha J(j)\big) \right],$$

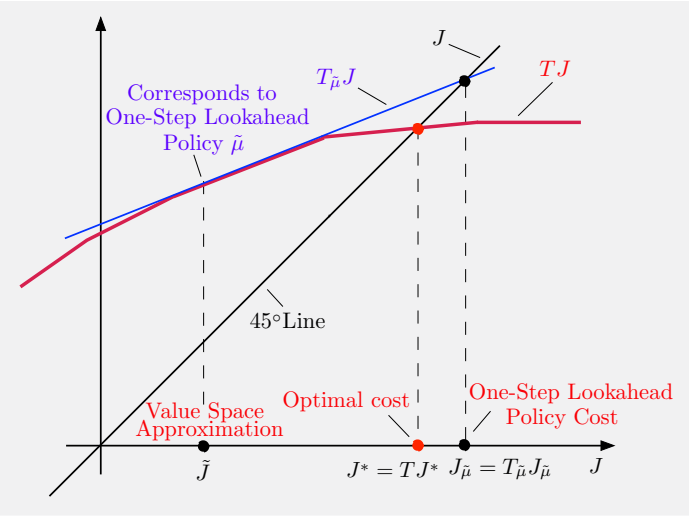


Figure 1.3.10 Geometric interpretation of approximation in value space and the one-step lookahead policy $\tilde{\mu}$ [cf. Eq. (1.37)]. Each policy μ defines the linear function $T_\mu J$ of J, given by Eq. (1.39), and TJ can be written as $TJ = \min_\mu T_\mu J$, while $\tilde{\mu}$ attains the minimum over μ of $T_\mu \tilde{J}$. Note that if there are finitely many policies, the components of TJ are piecewise linear. Approximation in value space with ℓ-step lookahead minimization can be similarly interpreted: we obtain $\tilde{\mu}$ with the construction in the figure, using $T^\ell \tilde{J}$ instead of $T\tilde{J}$.

$$(TJ)(2) = \min \left[\sum_{j=1}^{2} p_{2j}(u)\big(g(2,u,j) + \alpha J(j)\big), \sum_{j=1}^{2} p_{2j}(v)\big(g(2,v,j) + \alpha J(j)\big) \right].$$

Clearly, $(TJ)(1)$ and $(TJ)(2)$ are concave and piecewise linear (with two pieces; more generally, as many linear pieces as the number of controls). The concavity property holds in generality because $(TJ)(x)$ is the minimum of a collection of linear functions of J, one for each control $u \in U(x)$.

The operator notation simplifies algorithmic descriptions, derivations, and proofs related to DP, and facilitates its extensions to infinite horizon DP models beyond the ones that we have discussed in this section (see Chapter 5, the DP textbook [Ber12], and the abstract DP monograph [Ber18a]). In particular, using the operators T and T_μ, we can write the VI algorithm in the compact form

$$J_{k+1} = TJ_k,$$

while the one-step lookahead policy $\tilde{\mu}$ is characterized by the equation

$$T_{\tilde{\mu}}\tilde{J} = T\tilde{J};$$

cf. Figs. 1.3.9 and 1.3.10, respectively.

1.3.5 Infinite Horizon - Policy Iteration, Rollout, and Newton's Method

Another major class of infinite horizon algorithms is based on *policy iteration* (PI for short), which involves the repeated use of policy improvement and rollout (start from some policy and generate an improved policy). Figure 1.3.11 describes the method and indicates that each of its iterations consists of two phases:

(a) *Policy evaluation*, which computes the cost function J_μ. One possibility is to solve the corresponding Bellman equation

$$J_\mu(x) = E_w\Big\{g\big(x,\mu(x),w\big) + \alpha J_\mu\big(f(x,\mu(x),w)\big)\Big\}, \qquad \text{for all } x. \tag{1.40}$$

However, the value $J_\mu(x)$ for any x can also be computed by Monte Carlo simulation, by averaging over many randomly generated trajectories the cost of the policy starting from x. Other, more sophisticated possibilities include the use of specialized simulation-based methods, such as *temporal difference methods*, for which there is extensive literature (see e.g., the books [BeT96], [SuB98], [Ber12]).

(b) *Policy improvement*, which computes the improved rollout policy $\tilde\mu$ [in the sense that $J_{\tilde\mu}(x) \le J_\mu(x)$ for all x] using the one-step lookahead minimization

$$\tilde\mu(x) \in \arg\min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J_\mu\big(f(x,u,w)\big)\Big\}, \qquad \text{for all } x. \tag{1.41}$$

Thus PI generates a sequence of policies $\{\mu^k\}$, by obtaining μ^{k+1} through a policy improvement operation using J_{μ^k} in place of J_μ in Eq. (1.41), which is obtained through policy evaluation of the preceding policy μ^k using Eq. (1.40). Note that by using the abstract notation of Bellman operators, the PI algorithm can be written in the compact form

$$T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k},$$

where $\{\mu^k\}$ is the generated policy sequence.

We will discuss several different forms of PI in Chapter 5. We will argue there that PI forms the foundation for self-learning in RL, i.e., learning from data that is self-generated (from the system itself as it operates) rather than from data supplied from an external source.

Example 1.3.5 (Policy Iteration for Linear Quadratic Problems)

Consider the infinite horizon version of the linear quadratic problem discussed in Example 1.3.3, assuming a discount factor $\alpha < 1$. We will derive the form of PI starting from a linear base policy of the form

$$\mu^0(x) = L_0 x,$$

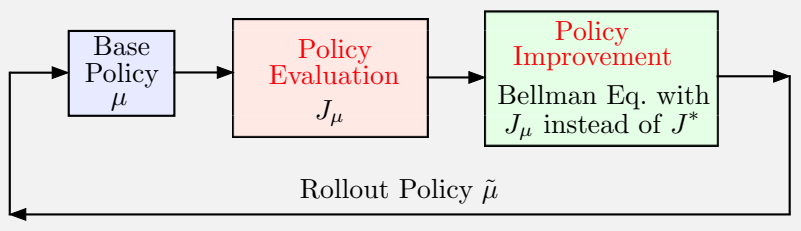


Figure 1.3.11 Schematic illustration of PI as repeated rollout. It generates a sequence of policies, with each policy μ in the sequence being the base policy that generates the next policy $\tilde{\mu}$ in the sequence as the corresponding rollout policy; cf. Eq. (1.41).

where L_0 is a scalar such that the closed loop system

$$x_{k+1} = (1 + bL_0)x_k + w_k, \tag{1.42}$$

is stable, i.e., $|1+bL_0| < 1$, or equivalently $-\frac{1}{b} < L_0 < 0$. This is necessary for the policy μ^0 to keep the state bounded and the corresponding costs $J_{\mu^0}(x)$ finite. We will see that the PI algorithm generates a sequence of linear stable policies.

We will now describe the policy evaluation and policy improvement phases for the starting policy μ^0. We can calculate J_{μ^0} by noting that it involves the uncontrolled closed loop system (1.42) and a quadratic cost function. Thus we expect that it has the form

$$J_{\mu^0}(x) = K_0 x^2 + \text{constant}, \tag{1.43}$$

and to calculate K_0 we will simply ignore the constant above, since it does not affect the minimization in the Bellman equation and in the VI operations. Equivalently, we will focus on the deterministic case ($w_k \equiv 0$), so that under the policy μ^0, we have

$$x_{k+1} = (1 + bL_0)x_k = (1 + bL_0)^k x_0,$$

and

$$J_{\mu^0}(x) = K_0 x^2 = \lim_{N\to\infty} \sum_{k=0}^{N-1} \alpha^k (1 + rL_0^2)(1 + bL_0)^{2k} x^2.$$

Thus, by cancelling x^2 from both sides above, we can calculate K_0 as

$$K_0 = \frac{1 + rL_0^2}{1 - \alpha(1 + bL_0)^2}. \tag{1.44}$$

In conclusion, the policy evaluation phase of PI for the starting linear policy $\mu^0(x) = L_0 x$ yields J_{μ^0} in the form (1.43)-(1.44). The policy improvement phase involves the quadratic minimization

$$\mu^1(x) \in \arg\min_u \left[x^2 + ru^2 + \alpha K_0 (x + bu)^2\right],$$

and after a straightforward calculation (setting to 0 the derivative with respect to u) yields μ^1 as the linear policy $\mu^1(x) = L_1 x$, where

$$L_1 = -\frac{\alpha b K_0}{r + \alpha b^2 K_0}. \tag{1.45}$$

It can also be verified that μ^1 is a stable policy. An intuitive way to get a sense of this is via the cost improvement property of PI: we have $J_{\mu^1}(x) \leq J_{\mu^0}(x)$ for all x, so $J_{\mu^1}(x)$ must be finite, which implies stability of μ^1.

The preceding calculation can be continued, so the PI algorithm yields the sequence of linear policies

$$\mu^k(x) = L_k x, \qquad k = 0, 1, \ldots, \tag{1.46}$$

where L_k is generated by the iteration

$$L_{k+1} = -\frac{\alpha b K_k}{r + \alpha b^2 K_k}, \tag{1.47}$$

[cf. Eq. (1.45)], with K_k given by

$$K_k = \frac{1 + r L_k^2}{1 - \alpha(1 + b L_k)^2}, \tag{1.48}$$

[cf. Eq. (1.44)]. The corresponding cost function sequence has the form $J_{\mu^k}(x) = K_k x^2$ for the deterministic problem where $w_k \equiv 0$, and the form

$$J_{\mu^k}(x) = K_k x^2 + \text{constant},$$

for the more general stochastic problem where w_k has zero mean but nonzero variance. It can be shown to converge to the optimal cost function J^*, while the generated sequence of linear policies $\{\mu^k\}$, where $\mu^k(x) = L_k x$, converges to the optimal policy. The convergence rate of the sequence $\{K_k\}$ has been shown to be *superlinear* (i.e., the ratio of the error of new iterate to the error of the previous iterate asymptotically tends to 0); see Exercise 1.5. We will discuss shortly this property, which stems from an interpretation of PI as a form of Newton's method.

In the deterministic case where $w_k \equiv 0$, PI is well-defined even in the undiscounted case where $\alpha = 1$. The generated policies are given by Eq. (1.46) with L_k given by

$$L_{k+1} = -\frac{b K_k}{r + b^2 K_k},$$

with

$$K_k = \frac{1 + r L_k^2}{1 - (1 + b L_k)^2},$$

[cf. Eqs. (1.47) and (1.48) for $\alpha = 1$]. Their cost functions are quadratic of the form $J_{\mu^k}(x) = K_k x^2$.

Our mathematical justification of the preceding analysis has been somewhat abbreviated, but it serves as a prototype for the analysis of more general multidimensional linear quadratic problems, and illustrates the PI process. Further discussion of the ideas underlying PI, including the policy improvement property, will be given in Chapter 5.

Relation to Newton's Method

The convergence rate of PI (in its exact form) is very fast in both theory and practice. In particular, it converges in a finite number of iterations for finite-state discounted problems, as we will see in Chapter 5. It also converges quadratically for linear quadratic problems (its iteration error is proportional to the square of the preceding iteration error) and superlinearly for other continuous-state problems.

An intuitive explanation is that PI can be viewed as a form of Newton's method for solving the Bellman equation

$$J(x) = \min_{u \in U(x)} E_w\Big\{ g(x,u,w) + \alpha J\big(f(x,u,w)\big)\Big\}$$

[cf. Eq. (1.25)], in the function space of cost functions $J(x)$. The analysis is complicated, but an insightful geometric interpretation is given in Fig. 1.3.12. The subject has a long history, for which we refer to the original papers by Kleinman [Klei68] for linear quadratic problems, and by Pollatschek and Avi-Itzhak [PoA69] for finite-state MDP and Markov game cases. Subsequent works include (among others) Hewer [Hew71], Puterman and Brumelle [PuB78], [PuB79], Saridis and Lee [SaL79] (following Rekasius [Rek64]), Beard [Bea95], Beard, Saridis, and Wen [BSW99], Santos and Rust [SaR04], Bokanowski, Maroso, and Zidani [BMZ09], Hylla [Hyl11], Magirou, Vassalos, and Barakitis [MVB20], and Kundu and Kunitsch [KuK21]. These papers discuss algorithmic variations under a variety of assumptions, and include superlinear convergence rate results.†

† Newton's method for solving a fixed point problem of the form $y = T(y)$, where y is an n-dimensional vector, operates as follows: At the current iterate y_k, we linearize T and find the solution y_{k+1} of the corresponding linear fixed point problem, obtained using a first order Taylor expansion:

$$y_{k+1} = T(y_k) + \frac{\partial T(y_k)}{\partial y}(y_{k+1} - y_k),$$

where $\partial T(y_k)/\partial y$ is the $n \times n$ Jacobian matrix of T evaluated at the n-dimensional vector y_k. The most commonly given convergence rate property of Newton's method is *quadratic convergence*. It states that near the solution y^*, we have

$$\|y_{k+1} - y^*\| = O\big(\|y_k - y^*\|^2\big),$$

where $\|\cdot\|$ is the Euclidean norm, and holds assuming the Jacobian matrix exists and is Lipschitz continuous (see [Ber16], Section 1.4). Qualitatively similar results hold under other assumptions. In particular a superlinear convergence statement (suitably modified to account for lack of differentiability of T) can be proved for problems where $T(y)$ has piecewise linear monotonically increasing and either concave or convex components. This is the version that is consistent with the geometric interpretation of PI given in Fig. 1.3.12.

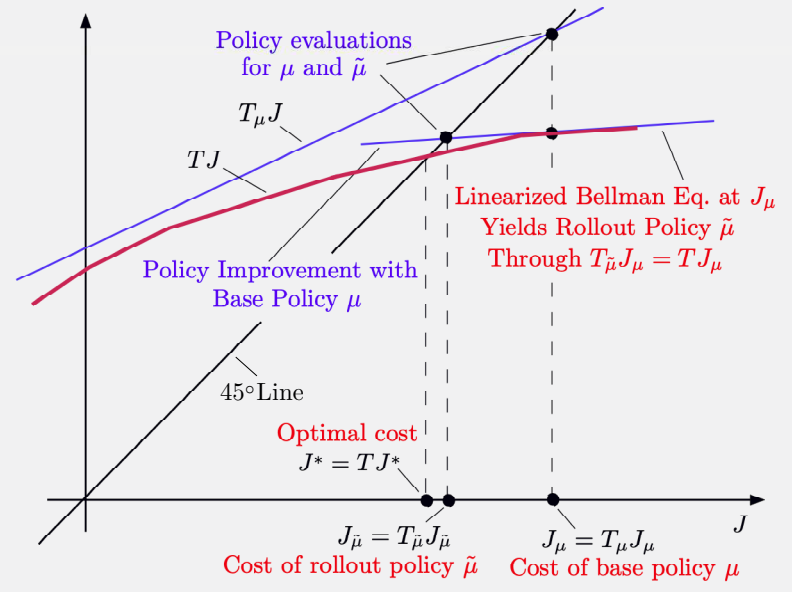


Figure 1.3.12 Geometric interpretation of PI and rollout. Each policy μ defines the linear function $T_\mu J$ of J, given by Eq. (1.39), and TJ is the function given by Eq. (1.38), which can also be written as $TJ = \min_\mu T_\mu J$. The figure shows a policy iteration starting from a base policy μ. It computes J_μ by policy evaluation (by solving the linear equation $J = T_\mu J$ as shown). It then performs a policy improvement using μ as the base policy to produce the rollout policy $\tilde{\mu}$ as shown: the cost function of the rollout policy, $J_{\tilde{\mu}}$, is obtained by solving the version of Bellman's equation that is linearized at the point J_μ, as in Newton's method. This process will be discussed further in Chapter 5.

Generally, *rollout with base policy μ can be viewed as a single iteration of Newton's method applied to the solution of the Bellman equation starting from J_μ*. This can be seen also from Fig. 1.3.12. In particular, given the base policy μ and its cost function J_μ, *the rollout algorithm first constructs a linearized version of Bellman's equation at J_μ (its linear approximation at J_μ), and then solves it to obtain $J_{\tilde{\mu}}$*. If the function TJ is nearly linear (has small "curvature,") the rollout policy performance $J_{\tilde{\mu}}(x)$ is very close to the optimal $J^*(x)$, even if the base policy μ is far from optimal. This explains the large cost improvements that are typically observed in practice with the rollout algorithm (see Fig. 1.3.12). Note that from Fig. 1.3.10, *approximation in value space with cost approximation $\tilde{J}$ can also be viewed as a single Newton iteration starting from $\tilde{J}$*.

An interesting question is how to compare the rollout performance for a given initial state, denote it by $\tilde{J}$, with the optimal performance, denote it by J^*, and with the base policy performance, denote it by J. Clearly, we would like $J - \tilde{J}$ to be large, but this is not the right way to look at cost improvement. The reason is that $J - \tilde{J}$ will be small if its upper bound,

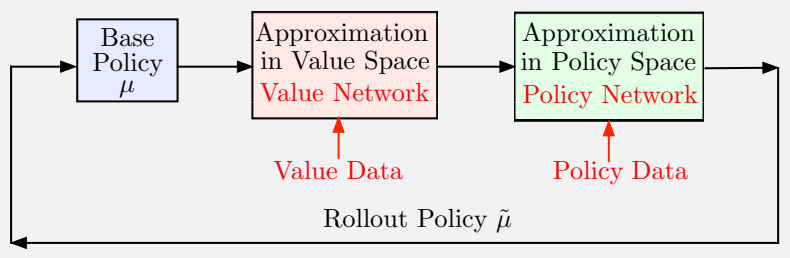


Figure 1.3.13 Schematic illustration of approximate PI. Either the policy evaluation and policy improvement phases (or both) are approximated with a value or a policy network, respectively. These could be neural networks, which are trained with (state, cost function value) data that is generated using the current base policy μ, and with (state, rollout policy control) data that is generated using the rollout policy $\tilde{\mu}$; see Chapters 4 and 5. Note that there are three different types of approximate implementation involving: 1) a value network but no policy network, or 2) a policy network but no value network, or 3) both a policy and a value network.

$J - J^*$, is small, i.e., if the base heuristic is close to optimal. What is important is that the error ratio

$$\frac{\tilde{J} - J^*}{J - J^*} \tag{1.49}$$

is small. Indeed, this ratio becomes smaller as $J - J^*$ approaches 0 because of the superlinear/quadratic convergence rate of Newton's method that underlies the rollout algorithm (cf. Fig. 1.3.12). Unfortunately, it is hard to evaluate this ratio, since we do not know J^*. On the other hand, we should not be underwhelmed if we observe a small performance improvement $J - \tilde{J}$: the reason may be that the base policy is already near-optimal, and in fact we may be doing very well in terms of the ratio (1.49).

Approximate Policy Iteration

Both policy evaluation and policy improvement can be approximated, possibly using neural networks, and training data collected through simulation, as we will discuss in Chapters 4 and 5; see Fig. 1.3.13. Other approximations include truncated rollout for policy evaluation. This is closely connected to a variant of PI called *optimistic*. Moreover, multistep lookahead may be used in place of one-step lookahead, and simplified minimization, based for example on multiagent rollout, may also be used. Let us also mention the possibility of a combined rollout and PI algorithm, whereby we use PI for on-line policy improvement of the base policy, by using data collected during the rollout process. This idea is relatively new and has not been tested extensively. It is described in Exercise 5.6 and in [Ber21b].

Note that the methods for policy evaluation and for policy improvement can be designed largely independently of each other. Moreover, there

are many exact and approximate methods for policy evaluation. In this book we will focus primarily on methods that evaluate J_μ using samples of pairs $\big(x, J_\mu(x)\big)$. We will not discuss alternatives, such as *temporal difference* and *aggregation* methods, which are based on the solution of projected equations that approximate Bellman's equation for J_μ; see the author's survey [Ber11b], and the textbook [Ber12], Chapters 6 and 7. They are popular and are supported by extensive theoretical analysis, but they are not central to our principal themes of rollout and distributed computation.

Finally, we should emphasize the important link between off-line and on-line PI-type algorithms. The final policy and policy evaluation obtained by off-line (exact or approximate) PI can be used as base policy and cost function approximation, respectively, for implementing on-line a truncated rollout policy that provides both on-line policy improvement and also allows for on-line replanning. This type of scheme is largely responsible for the success of the AlphaGo/AlphaZero programs, as well as Tesauro's 1996 backgammon program. The base policies in both cases were obtained via sophisticated neural network training, but play at levels below top humans. However, the one-step or multistep lookahead policies that are used for on-line play perform much better than the best humans. We noted earlier in Section 1.1 this significant "policy improvement by on-line play" phenomenon: it is simply the improvement due to a single on-line policy iteration.

1.4 EXAMPLES, VARIATIONS, AND SIMPLIFICATIONS

In this section we provide a few examples that illustrate problem formulation techniques, exact and approximate solution methods, and adaptations of the basic DP algorithm to various contexts. We refer to DP textbooks for extensive additional discussions of modeling and problem formulation techniques (see e.g, the many examples that can be found in the author's DP and RL textbooks [Ber12], [Ber17a], [Ber19a], as well as in the neuro-dynamic programming monograph [BeT96]).

An important fact to keep in mind is that there are many ways to model a given practical problem in terms of DP, and that there is no unique choice for state and control variables. This will be brought out by the examples in this section, and is facilitated by the generality of DP: its basic algorithmic principles apply for arbitrary state, control, and disturbance spaces, and system and cost functions.

1.4.1 A Few Words About Modeling

In practice, optimization problems seldom come neatly packaged as mathematical problems that can be solved by DP/RL or some other methodology. Generally, a practical problem is a prime candidate for a DP formulation

if it involves multiple sequential decisions separated by the collection of information that can enhance the effectiveness of future decisions.

However, there are other types of problems that can be fruitfully formulated by DP. These include the entire class of deterministic problems, where there is no information to be collected: all the information needed in a deterministic problem is either known or can be predicted from the problem data that is available at time 0. Moreover, for deterministic problems there is a plethora of non-DP methods, such as linear, nonlinear, and integer programming, random and nonrandom search, discrete optimization heuristics, etc. Still, however, the use of RL methods for deterministic optimization is a major subject in this book, which will be discussed in Chapters 2 and 3. We will argue there that rollout, when suitably applied, can improve substantially the performance of other heuristic or suboptimal methods, however derived. Moreover, we will see that often for discrete optimization problems the DP sequential structure is introduced artificially, with the aim to facilitate the use of approximate DP/RL methods.

There are also problems that fit quite well into the sequential structure of DP, but can be fruitfully addressed by RL methods that do not have a fundamental connection with DP. An important case in point is *policy gradient and policy search* methods, which will be considered only peripherally in this book. Here the policy of the problem is parametrized by a set of parameters, so that the cost of the policy becomes a function of these parameters, and can be optimized by non-DP methods such as gradient or random search-based suboptimal approaches. This is approximation in policy space approach, which will be discussed further in Chapter 2; see also Section 5.7 of the RL book [Ber19a] and the end-of-chapter references.

As a guide for formulating optimal control problems in a manner that is suitable for a DP solution the following two-stage process is suggested:

(a) Identify the controls/decisions u_k and the times k at which these controls are applied. Usually this step is fairly straightforward. However, in some cases there may be some choices to make. For example in deterministic problems, where the objective is to select an optimal sequence of controls $\{u_0, \ldots, u_{N-1}\}$, one may lump multiple controls to be chosen together, e.g., view the pair (u_0, u_1) as a single choice. This is usually not possible in stochastic problems, where distinct decisions are differentiated by the information/feedback available when making them.

(b) Select the states x_k. The basic guideline here is that x_k should encompass *all the information that is relevant for future optimization*, i.e., the information that is known to the controller at time k and can be used with advantage in choosing u_k. In effect, at time k *the state x_k should separate the past from the future*, in the sense that anything that has happened in the past (states, controls, and disturbances from stages prior to stage k) is irrelevant to the choices

of future controls as long we know x_k. Sometimes this is described by saying that the state should have a "Markov property" to express an analogy with states of Markov chains, where (by definition) the conditional probability distribution of future states depends on the past history of the chain only through the present state.

The control and state selection may also have to be refined or specialized in order to enhance the application of known results and algorithms. This includes the choice of a finite or an infinite horizon, and the availability of good base policies or heuristics in the context of rollout.

Note that there may be multiple possibilities for selecting the states, because information may be packaged in several different ways that are equally useful from the point of view of control. It may thus be worth considering alternative ways to choose the states; for example try to use states that minimize the dimensionality of the state space. For a trivial example that illustrates the point, if a quantity x_k qualifies as state, then (x_{k-1}, x_k) also qualifies as state, since (x_{k-1}, x_k) contains all the information contained within x_k that can be useful to the controller when selecting u_k. However, using (x_{k-1}, x_k) in place of x_k, gains nothing in terms of optimal cost while complicating the DP algorithm that would have to be executed over a larger space.

The concept of a *sufficient statistic*, which refers to a quantity that summarizes all the essential content of the information available to the controller, may be useful in providing alternative descriptions of the state space. An important paradigm is problems involving *partial* or *imperfect* state information, where x_k evolves over time but is not fully accessible for measurement (for example, x_k may be the position/velocity vector of a moving vehicle, but we may obtain measurements of just the position). If I_k is the collection of all measurements and controls up to time k (the *information vector*), it is correct to use I_k as state in a reformulated DP problem that involves perfect state observation. However, a better alternative may be to use as state the conditional probability distribution $P_k(x_k \mid I_k)$, called *belief state*, which (as it turns out) subsumes all the information that is useful for the purposes of choosing a control. On the other hand, the belief state $P_k(x_k \mid I_k)$ is an infinite-dimensional object, whereas I_k may be finite dimensional, so the best choice may be problem-dependent. Still, in either case, the stochastic DP algorithm applies, with the sufficient statistic [whether I_k or $P_k(x_k \mid I_k)$] playing the role of the state; see Section 1.4.4.

1.4.2 Problems with a Termination State

Many DP problems of interest involve a *termination state*, i.e., a state t that is cost-free and absorbing in the sense that for all k,

$$g_k(t, u_k, w_k) = 0, \quad f_k(t, u_k, w_k) = t, \qquad \text{for all } w_k \text{ and } u_k \in U_k(t).$$

Thus the control process essentially terminates upon reaching t, even if this happens before the end of the horizon. One may reach t by choice if a special stopping decision is available, or by means of a random transition from another state. Problems involving games, such as chess, Go, backgammon, and others involve a termination state (the end of the game).†

Generally, when it is known that an optimal policy will reach the termination state with certainty within at most some given number of stages N, the DP problem can be formulated as an N-stage horizon problem, with a very large termination cost for the nontermination states.‡ The reason is that even if the termination state t is reached at a time $k < N$, we can extend our stay at t for an additional $N - k$ stages at no additional cost, so the optimal policy will still be optimal, since it will not incur the large termination cost at the end of the horizon. An example is the multi-vehicle problem of Example 1.2.3: we reach the termination state once all tasks have been performed, but the number of stages for this to occur is unknown and depends on the policy. However, a bound of the required time for termination by an optimal policy can be easily calculated.

Example 1.4.1 (Parking)

A driver is looking for inexpensive parking on the way to his destination. The parking area contains N spaces, numbered $0, \ldots, N-1$, and a garage following space $N-1$. The driver starts at space 0 and traverses the parking spaces sequentially, i.e., from space k he goes next to space $k+1$, etc. Each parking space k costs $c(k)$ and is free with probability $p(k)$ independently of whether other parking spaces are free or not. If the driver reaches the last parking space $N-1$ and does not park there, he must park at the garage, which costs C. The driver can observe whether a parking space is free only when he reaches it, and then, if it is free, he makes a decision to park in that space or not to park and check the next space. The problem is to find the minimum expected cost parking policy.

We formulate the problem as a DP problem with N stages, corresponding to the parking spaces, and an artificial termination state t that corresponds to having parked; see Fig. 1.4.1. At each stage $k = 1, \ldots, N-1$, we have three states: the artificial termination state, and the two states F and $\overline{F}$, corresponding to space k being free or taken, respectively. At stage 0, we have only two states, F and $\overline{F}$, and at the final stage there is only one state, the termination state t. The decision/control is to park or continue at state F

† Games often involve two players/decision makers, in which case they can be addressed by suitably modified exact or approximate DP algorithms. The DP algorithm that we have discussed in this chapter involves a single decision maker, but can be used to find an optimal policy for one player against a fixed and known policy of the other player.

‡ When an upper bound on the number of stages to termination is not known, the problem may be formulated as an infinite horizon problem of the stochastic shortest path type, which we will discuss in Chapter 5.

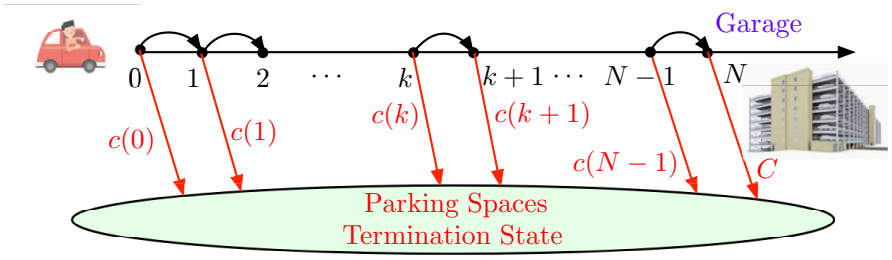


Figure 1.4.1 Cost structure of the parking problem. The driver may park at space $k = 0, 1, \ldots, N-1$ at cost $c(k)$, if the space is free, or continue to the next space $k+1$ at no cost. At space N (the garage) the driver must park at cost C.

[there is no choice at states $\overline{F}$ and state t]. From location k, the termination state t is reached at cost $c(k)$ when a parking decision is made (assuming location k is free). Otherwise, the driver continues to the next state at no cost. At stage N, the driver must park at cost C.

Let us now derive the form of the DP algorithm, denoting:

$J_k^*(F)$: The optimal cost-to-go upon arrival at a space k that is free.

$J_k^*(\overline{F})$: The optimal cost-to-go upon arrival at a space k that is taken.

$J_k^*(t)$: The cost-to-go of the "parked"/termination state t.

The DP algorithm for $k = 0, \ldots, N-1$ takes the form

$$J_k^*(F) = \begin{cases} \min\big[c(k),\, p(k+1)J_{k+1}^*(F) + \big(1-p(k+1)\big)J_{k+1}^*(\overline{F})\big] & \text{if } k < N-1, \\ \min\big[c(N-1),\, C\big] & \text{if } k = N-1, \end{cases}$$

$$J_k^*(\overline{F}) = \begin{cases} p(k+1)J_{k+1}^*(F) + \big(1-p(k+1)\big)J_{k+1}^*(\overline{F}) & \text{if } k < N-1, \\ C & \text{if } k = N-1, \end{cases}$$

for the states other than the termination state t, while for t we have

$$J_k^*(t) = 0, \qquad k = 1, \ldots, N.$$

The minimization above corresponds to the two choices (park or not park) at the states F that correspond to a free parking space.

While this algorithm is easily executed, it can be written in a simpler and equivalent form. This can be done by introducing the scalars

$$\hat{J}_k = p(k)J_k^*(F) + \big(1-p(k)\big)J_k^*(\overline{F}), \qquad k = 0, \ldots, N-1,$$

which can be viewed as the optimal expected cost-to-go upon arriving at space k but *before verifying its free or taken status*. Indeed, from the preceding DP algorithm, we have

$$\hat{J}_{N-1} = p(N-1)\min\big[c(N-1),\, C\big] + \big(1-p(N-1)\big)C,$$

$$\hat{J}_k = p(k)\min\big[c(k),\, \hat{J}_{k+1}\big] + \big(1-p(k)\big)\hat{J}_{k+1}, \qquad k = 0, \ldots, N-2.$$

From this algorithm we can also obtain the optimal parking policy:

Park at space $k = 0, \ldots, N-1$ if it is free and we have $c(k) \leq \hat{J}_{k+1}$.

1.4.3 State Augmentation, Time Delays, Forecasts, and Uncontrollable State Components

In practice, we are often faced with situations where some of the assumptions of our stochastic optimal control problem are violated. For example, the disturbances may involve a complex probabilistic description that may create correlations that extend across stages, or the system equation may include dependences on controls applied in earlier stages, which affect the state with some delay.

Generally, in such cases the problem can be reformulated into our DP problem format through a technique, which is called *state augmentation* because it typically involves the enlargement of the state space. The general guideline in state augmentation is to *include in the enlarged state at time k all the information that is known to the controller at time k and can be used with advantage in selecting u_k*. State augmentation allows the treatment of time delays in the effects of control on future states, correlated disturbances, forecasts of probability distributions of future disturbances, and many other complications. We note, however, that state augmentation often comes at a price: the reformulated problem may have a very complex state space. We provide some examples.

Time Delays

In some applications the system state x_{k+1} depends not only on the preceding state x_k and control u_k, but also on earlier states and controls. Such situations can be handled by expanding the state to include an appropriate number of earlier states and controls.

As an example, assume that there is at most a single stage delay in the state and control; i.e., the system equation has the form

$$x_{k+1} = f_k(x_k, x_{k-1}, u_k, u_{k-1}, w_k), \qquad k = 1, \ldots, N-1, \tag{1.50}$$

$$x_1 = f_0(x_0, u_0, w_0).$$

If we introduce additional state variables y_k and s_k, and we make the identifications $y_k = x_{k-1}$, $s_k = u_{k-1}$, the system equation (1.50) yields

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \\ s_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, y_k, u_k, s_k, w_k) \\ x_k \\ u_k \end{pmatrix}. \tag{1.51}$$

By defining $\tilde{x}_k = (x_k, y_k, s_k)$ as the new state, we have

$$\tilde{x}_{k+1} = \tilde{f}_k(\tilde{x}_k, u_k, w_k),$$

where the system function $\tilde{f}_k$ is defined from Eq. (1.51).

By using the preceding equation as the system equation and by expressing the cost function in terms of the new state, the problem is reduced to a problem without time delays. Naturally, the control u_k should now depend on the new state $\tilde{x}_k$, or equivalently a policy should consist of functions μ_k of the current state x_k, as well as the preceding state x_{k-1} and the preceding control u_{k-1}.

When the DP algorithm for the reformulated problem is translated in terms of the variables of the original problem, it takes the form

$$J_N^*(x_N) = g_N(x_N),$$

$$\begin{aligned} &J_{N-1}(x_{N-1}, x_{N-2}, u_{N-2}) \\ &\quad = \min_{u_{N-1} \in U_{N-1}(x_{N-1})} E_{w_{N-1}} \Big\{ g_{N-1}(x_{N-1}, u_{N-1}, w_{N-1}) \\ &\qquad\qquad + J_N^*\big(f_{N-1}(x_{N-1}, x_{N-2}, u_{N-1}, u_{N-2}, w_{N-1})\big) \Big\}, \end{aligned}$$

$$\begin{aligned} J_k^*(x_k, x_{k-1}, u_{k-1}) = \min_{u_k \in U_k(x_k)} E_{w_k} \Big\{ & g_k(x_k, u_k, w_k) \\ & + J_{k+1}^*\big(f_k(x_k, x_{k-1}, u_k, u_{k-1}, w_k), x_k, u_k\big) \Big\}, \quad k = 1, \ldots, N-2, \end{aligned}$$

$$J_0^*(x_0) = \min_{u_0 \in U_0(x_0)} E_{w_0} \Big\{ g_0(x_0, u_0, w_0) + J_1^*\big(f_0(x_0, u_0, w_0), x_0, u_0\big) \Big\}.$$

Similar reformulations are possible when time delays appear in the cost or the control constraints; for example, in the case where the cost has the form

$$E\left\{ g_N(x_N, x_{N-1}) + g_0(x_0, u_0, w_0) + \sum_{k=1}^{N-1} g_k(x_k, x_{k-1}, u_k, w_k) \right\}.$$

The extreme case of time delays in the cost arises in the nonadditive form

$$E\big\{ g_N(x_N, x_{N-1}, \ldots, x_0, u_{N-1}, \ldots, u_0, w_{N-1}, \ldots, w_0) \big\}.$$

Then, the problem can be reduced to the standard problem format, by using as augmented state

$$\tilde{x}_k = \big(x_k, x_{k-1}, \ldots, x_0, u_{k-1}, \ldots, u_0, w_{k-1}, \ldots, w_0\big)$$

and $E\big\{g_N(\tilde{x}_N)\big\}$ as reformulated cost. Policies consist of functions μ_k of the present and past states $x_k, \ldots, x_0$, the past controls $u_{k-1}, \ldots, u_0$, and the past disturbances $w_{k-1}, \ldots, w_0$. Naturally, we must assume that the past disturbances are known to the controller. Otherwise, we are faced with a problem where the state is imprecisely known to the controller, which will be discussed in the next section.

Forecasts

Consider a situation where at time k the controller has access to a forecast y_k that results in a reassessment of the probability distribution of the subsequent disturbance w_k and, possibly, future disturbances. For example, y_k may be an exact prediction of w_k or an exact prediction that the probability distribution of w_k is a specific one out of a finite collection of distributions. Forecasts of interest in practice are, for example, probabilistic predictions on the state of the weather, the interest rate for money, and the demand for inventory. Generally, forecasts can be handled by introducing additional state variables corresponding to the information that the forecasts provide. We will illustrate the process with a simple example.

Assume that at the beginning of each stage k, the controller receives an accurate prediction that the next disturbance w_k will be selected according to a particular probability distribution out of a given collection of distributions $\{P_1, \ldots, P_m\}$; i.e., if the forecast is i, then w_k is selected according to P_i. The a priori probability that the forecast will be i is denoted by p_i and is given.

The forecasting process can be represented by means of the equation

$$y_{k+1} = \xi_k,$$

where y_{k+1} can take the values $1, \ldots, m$, corresponding to the m possible forecasts, and ξ_k is a random variable taking the value i with probability p_i. The interpretation here is that when ξ_k takes the value i, then w_{k+1} will occur according to the distribution P_i.

By combining the system equation with the forecast equation $y_{k+1} = \xi_k$, we obtain an augmented system given by

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, u_k, w_k) \\ \xi_k \end{pmatrix}.$$

The new state and disturbance are

$$\tilde{x}_k = (x_k, y_k), \qquad \tilde{w}_k = (w_k, \xi_k).$$

The probability distribution of $\tilde{w}_k$ is determined by the distributions P_i and the probabilities p_i, and depends explicitly on $\tilde{x}_k$ (via y_k) but not on the prior disturbances.

Thus, by suitable reformulation of the cost, the problem can be cast as a stochastic DP problem. Note that the control applied depends on both the current state and the current forecast. The DP algorithm takes the form

$$J_N^*(x_N, y_N) = g_N(x_N),$$

$$J_k^*(x_k, y_k) = \min_{u_k \in U_k(x_k)} E_{w_k} \Bigg\{ g_k(x_k, u_k, w_k) + \sum_{i=1}^{m} p_i J_{k+1}^*\big(f_k(x_k, u_k, w_k), i\big) \;\Big|\; y_k \Bigg\}, \tag{1.52}$$

where y_k may take the values $1, \ldots, m$, and the expected value over w_k is taken with respect to the distribution P_{y_k}.

Note that the preceding formulation admits several extensions. One example is the case where forecasts can be influenced by the control action (e.g., pay extra for a more accurate forecast), and may involve several future disturbances. However, the price for these extensions is increased complexity of the corresponding DP algorithm.

Problems with Uncontrollable State Components

In many problems of interest the natural state of the problem consists of several components, some of which cannot be affected by the choice of control. In such cases the DP algorithm can be simplified considerably, and be executed over the controllable components of the state.

As an example, let the state of the system be a composite (x_k, y_k) of two components x_k and y_k. The evolution of the main component, x_k, is affected by the control u_k according to the equation

$$x_{k+1} = f_k(x_k, y_k, u_k, w_k),$$

where the distribution $P_k(w_k \mid x_k, y_k, u_k)$ is given. The evolution of the other component, y_k, is governed by a given conditional distribution $P_k(y_k \mid x_k)$ and cannot be affected by the control, except indirectly through x_k. One is tempted to view y_k as a disturbance, but there is a difference: y_k is observed by the controller before applying u_k, while w_k occurs after u_k is applied, and indeed w_k may probabilistically depend on u_k.

It turns out that we can formulate a DP algorithm that is executed over the controllable component of the state, with the dependence on the uncontrollable component being "averaged out" as in the preceding example (see also the parking Example 1.4.1). In particular, let $J_k^*(x_k, y_k)$ denote the optimal cost-to-go at stage k and state (x_k, y_k), and define

$$\hat{J}_k(x_k) = E_{y_k}\big\{ J_k^*(x_k, y_k) \mid x_k \big\}.$$

Note that the preceding expression can be interpreted as an "average cost-to-go" at x_k (averaged over the values of the uncontrollable component y_k). Then, similar to the preceding parking example, a DP algorithm that

generates $\hat{J}_k(x_k)$ can be obtained, and has the following form:

$$\hat{J}_k(x_k) = E_{y_k}\Big\{ \min_{u_k \in U_k(x_k,y_k)} E_{w_k}\Big\{ g_k(x_k,y_k,u_k,w_k) + \hat{J}_{k+1}\big(f_k(x_k,y_k,u_k,w_k)\big) \mid x_k,y_k,u_k\Big\} \Big| \, x_k\Big\}. \tag{1.53}$$

This is a consequence of the calculation

$$\begin{aligned}\hat{J}_k(x_k) &= E_{y_k}\big\{ J_k^*(x_k,y_k) \mid x_k\big\} \\ &= E_{y_k}\Big\{ \min_{u_k \in U_k(x_k,y_k)} E_{w_k,x_{k+1},y_{k+1}}\big\{ g_k(x_k,y_k,u_k,w_k) \\ &\qquad + J_{k+1}^*(x_{k+1},y_{k+1}) \mid x_k,y_k,u_k\big\} \mid x_k\Big\} \\ &= E_{y_k}\Big\{ \min_{u_k \in U_k(x_k,y_k)} E_{w_k,x_{k+1}}\Big\{ g_k(x_k,y_k,u_k,w_k) \\ &\qquad + E_{y_{k+1}}\big\{ J_{k+1}^*(x_{k+1},y_{k+1}) \mid x_{k+1}\big\} \mid x_k,y_k,u_k\Big\} \Big| \, x_k\Big\}.\end{aligned}$$

Note that the minimization in the right-hand side of the preceding equation must still be performed for all values of the full state (x_k, y_k) in order to yield an optimal control law as a function of (x_k, y_k). Nonetheless, the equivalent DP algorithm (1.53) has the advantage that it is executed over a significantly reduced state space. Later, when we consider approximation in value space, we will find that it is often more convenient to approximate $\hat{J}_k(x_k)$ than to approximate $J_k^*(x_k, y_k)$; see the following discussions of forecasts and of the game of tetris.

As an example, consider the augmented state resulting from the incorporation of forecasts, as described earlier. Then, the forecast y_k represents an uncontrolled state component, so that the DP algorithm can be simplified as in Eq. (1.53). In particular, assume that the forecast y_k can take values $i = 1, \ldots, m$ with probability p_i. Then, by defining

$$\hat{J}_k(x_k) = \sum_{i=1}^{m} p_i J_k^*(x_k,i), \qquad k = 0,1,\ldots,N-1,$$

and $\hat{J}_N(x_N) = g_N(x_N)$, we have using Eq. (1.52),

$$\hat{J}_k(x_k) = \sum_{i=1}^{m} p_i \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ g_k(x_k,u_k,w_k) + \hat{J}_{k+1}\big(f_k(x_k,u_k,w_k)\big) \mid y_k = i\Big\},$$

which is executed over the space of x_k rather than x_k *and* y_k. Note that this is a simpler algorithm to approximate than the one of Eq. (1.52).

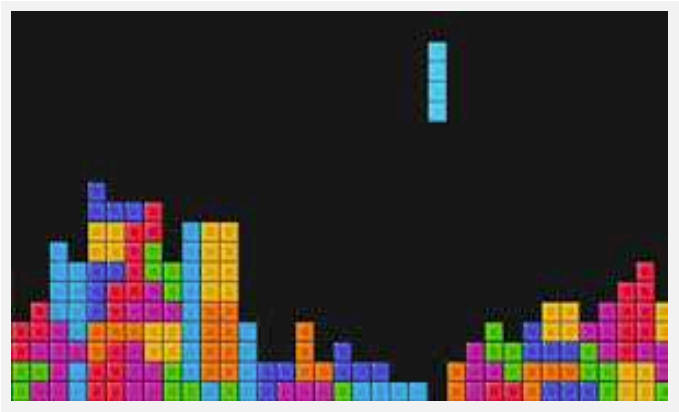


Figure 1.4.2 Illustration of a tetris board.

Uncontrollable state components often occur in arrival systems, such as queueing, where action must be taken in response to a random event (such as a customer arrival) that cannot be influenced by the choice of control. Then the state of the arrival system must be augmented to include the random event, but the DP algorithm can be executed over a smaller space, as per Eq. (1.53). Here is an example of this type.

Example 1.4.2 (Tetris)

Tetris is a popular video game played on a two-dimensional grid. Each square in the grid can be full or empty, making up a "wall of bricks" with "holes" and a "jagged top" (see Fig. 1.4.2). The squares fill up as blocks of different shapes fall from the top of the grid and are added to the top of the wall. As a given block falls, the player can move horizontally and rotate the block in all possible ways, subject to the constraints imposed by the sides of the grid and the top of the wall. The falling blocks are generated independently according to some probability distribution, defined over a finite set of standard shapes. The game starts with an empty grid and ends when a square in the top row becomes full and the top of the wall reaches the top of the grid. When a row of full squares is created, this row is removed, the bricks lying above this row move one row downward, and the player scores a point. The player's objective is to maximize the score attained (total number of rows removed) up to termination of the game, whichever occurs first.

We can model the problem of finding an optimal tetris playing strategy as a finite horizon stochastic DP problem, with very long horizon. The state consists of two components:

(1) The board position, i.e., a binary description of the full/empty status of each square, denoted by x.

(2) The shape of the current falling block, denoted by y.

The control, denoted by u, is the horizontal positioning and rotation applied to the falling block. There is also an additional termination state which is cost-free. Once the state reaches the termination state, it stays there with no change in score. Moreover there is a very large amount added to the score when the end of the horizon is reached without the game having terminated.

The shape y is generated according to a probability distribution $p(y)$, independently of the control, so it can be viewed as an uncontrollable state component. The DP algorithm (1.53) is executed over the space of board positions x and has the intuitive form

$$\hat{J}_k(x) = \sum_y p(y) \max_u \Big[g(x,y,u) + \hat{J}_{k+1}\big(f(x,y,u)\big) \Big], \qquad \text{for all } x, \tag{1.54}$$

where

$g(x,y,u)$ is the number of points scored (rows removed),

$f(x,y,u)$ is the next board position (or termination state),

when the state is (x,y) and control u is applied, respectively. The DP algorithm (1.54) assumes a finite horizon formulation of the problem.

Alternatively, we may consider an undiscounted infinite horizon formulation, involving a termination state (i.e., a stochastic shortest path problem). The "reduced" form of Bellman's equation, which corresponds to the DP algorithm (1.54), has the form

$$\hat{J}(x) = \sum_y p(y) \max_u \Big[g(x,y,u) + \hat{J}\big(f(x,y,u)\big) \Big], \qquad \text{for all } x.$$

The value $\hat{J}(x)$ can be interpreted as an "average score" at x (averaged over the values of the uncontrollable block shapes y).

Finally, let us note that despite the simplification achieved by eliminating the uncontrollable portion of the state, the number of states x is still enormous, and the problem can only be addressed by suboptimal methods, which will be discussed later in this book.†

1.4.4 Partial State Information and Belief States

We have assumed so far that the controller has access to the exact value of the current state x_k, so a policy consists of a sequence of functions $\mu_k(\cdot)$, $k = 0, \ldots, N-1$. However, in many practical settings this assumption is unrealistic, because some components of the state may be inaccessible for observation, the sensors used to measure them may be inaccurate, or the cost of more accurate observations may be prohibitive.

† Tetris has received a lot of attention as a challenging testbed for RL algorithms over a period spanning 20 years (1996-2015), starting with the papers [TsV96] and [BeI96], and ending with the papers [GGS13], [SGG15], which contain many references to related works in the intervening years.

Often in such situations the controller has access to only some of the components of the current state, and the corresponding observations may also be corrupted by stochastic uncertainty. For example in three-dimensional motion problems, the state may consist of the six-tuple of position and velocity components, but the observations may consist of noise-corrupted radar measurements of the three position components. This gives rise to problems of *partial* or *imperfect* state information, which have received a lot of attention in the optimization and artificial intelligence literature (see e.g., [Ber17a], [RuN16]; these problems are also popularly referred to with the acronym POMDP for *partially observed Markovian decision problem*).

Generally, there are DP algorithms for solving a POMDP exactly, although the computation necessary for this is typically intractable in practice. The most common approach is to replace the state x_k with a *belief state*, which we will denote by b_k. It is the probability distribution of x_k given all the observations that have been obtained by the controller up to time k, and it can serve as "state" in an appropriate DP algorithm. The belief state can in principle be computed and updated by a variety of methods that are based on Bayes' rule, such as *Kalman filtering* (see [AnM79], [KuV86], [Kri16], [ChC17]) and *particle filtering* (see [GSS93], [DoJ09], [Can16], [Kri16]).

In problems where the state x_k can take a finite but large number of values, say n, the belief states comprise an n-dimensional simplex, so discretization becomes problematic. As a result, alternative suboptimal solution methods are often used in POMDP. Some of these methods will be described in future chapters.

Example 1.4.3 (Bidirectional Parking)

Let us consider a more complex version of the parking problem of Example 1.4.1. As in that example, a driver is looking for inexpensive parking on the way to his destination, along a line of N parking spaces with a garage at the end. The difference is that the driver can move in either direction, rather than just forward towards the garage. In particular, at space i, the driver can park at cost $c(i)$ if i is free, can move to $i-1$ at a cost t_i^- or can move to $i+1$ at a cost t_i^+. Moreover, the driver records and remembers the free/taken status of the spaces previously visited and may return to any of these spaces; see Fig. 1.4.3.

Let us assume that the probability $p(i)$ of a space i being free changes over time, i.e., a space found free (or taken) at a given visit may get taken (or become free, respectively) by the time of the next visit. The initial probabilities $p(i)$, before visiting any spaces, are known, and the mechanism by which these probabilities change over time is also known to the driver. As an example, we may assume that at each time stage, $p(i)$ increases by a certain known factor with some probability ξ and decreases by another known factor with the complementary probability $1-\xi$.

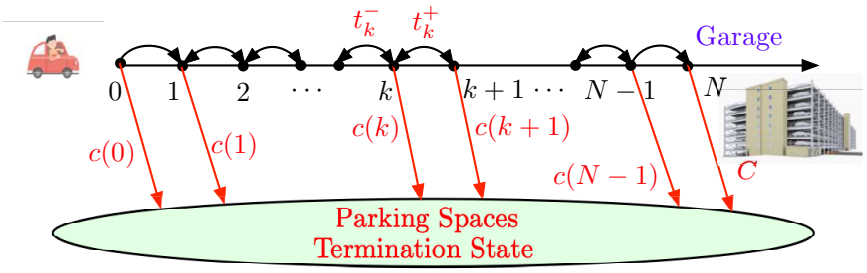


Figure 1.4.3 Cost structure and transitions of the bidirectional parking problem. The driver may park at space $k = 0, 1, \ldots, N-1$ at cost $c(k)$, if the space is free, can move to $k-1$ at cost t_k^-, or can move to $k+1$ at cost t_k^+. At space N (the garage) the driver must park at cost C.

Here the belief state is the vector of current probabilities

$$\big(p(0), \ldots, p(N-1)\big),$$

and it can be updated with a simple algorithm at each time based on the new observation: the free/taken status of the space visited at that time.

Despite their inherent computational difficulty, it turns out that conceptually, partial state information problems are no different than the perfect state information problems we have been addressing so far. In fact by various reformulations, we can reduce a partial state information problem to one with perfect state information. Once this is done, it is possible to state an exact DP algorithm that is defined over the set of belief states. This algorithm has the form

$$J_k^*(b_k) = \min_{u_k \in U_k} \Big[\hat{g}_k(b_k, u_k) + E_{z_{k+1}} \Big\{ J_{k+1}^*\big(F_k(b_k, u_k, z_{k+1})\big) \Big\} \Big], \qquad (1.55)$$

where:

$J_k^*(b_k)$ denotes the optimal cost-to-go starting from belief state b_k at stage k.

U_k is the control constraint set at time k (since the state x_k is unknown at stage k, U_k must be independent of x_k).

$\hat{g}_k(b_k, u_k)$ denotes the expected stage cost of stage k. It is calculated as the expected value of the stage cost $g_k(x_k, u_k, w_k)$, with the joint distribution of (x_k, w_k) determined by the belief state b_k and the distribution of w_k.

$F_k(b_k, u_k, z_{k+1})$ denotes the belief state at the next stage, given that the current belief state is b_k, control u_k is applied, and observation z_{k+1} is received following the application of u_k:

$$b_{k+1} = F_k(b_k, u_k, z_{k+1}). \qquad (1.56)$$

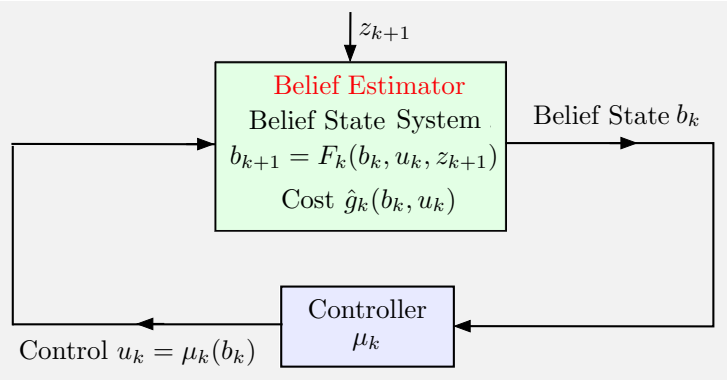


Figure 1.4.4 Schematic illustration of the view of an imperfect state information problem as one of perfect state information, whose state is the belief state b_k, i.e., the conditional probability distribution of x_k given all the observations up to time k. The observation z_{k+1} plays the role of the stochastic disturbance. The function F_k is a sequential estimator that updates the current belief state b_k.

This is the system equation for a perfect state information problem with state b_k, control u_k, "disturbance" z_{k+1}, and cost per stage $\hat{g}_k(b_k, u_k)$. The function F_k is viewed as a sequential *belief estimator*, which updates the current belief state b_k based on the new observation z_{k+1}. It is given by either an explicit formula or an algorithm (such as Kalman filtering or particle filtering) that is based on the probability distribution of z_k and the use of Bayes' rule.

The expected value $E_{z_{k+1}}\{\cdot\}$ is taken with respect to the distribution of z_{k+1}, given b_k and u_k. Note that z_{k+1} is random, and its distribution depends on x_k and u_k, so the expected value

$$E_{z_{k+1}}\Big\{J^*_{k+1}\big(F_k(b_k, u_k, z_{k+1})\big)\Big\}$$

in Eq. (1.55) is a function of b_k and u_k.

The algorithm (1.55) is just the ordinary DP algorithm for the perfect state information problem shown in Fig. 1.4.4. It involves the system/belief estimator (1.56) and the cost per stage $\hat{g}_k(b_k, u_k)$. Note that since b_k takes values in a continuous space, the algorithm (1.55) can only be executed approximately, using approximation in value space methods.

We refer to the textbook [Ber17a], Chapter 4, for a detailed derivation of the DP algorithm (1.55), and to the monograph [BeS78] for a mathematical treatment that applies to infinite state, control, and disturbance spaces as well.

1.4.5 Multiagent Problems and Multiagent Rollout

Let us consider the discounted infinite horizon problem and a special struc-

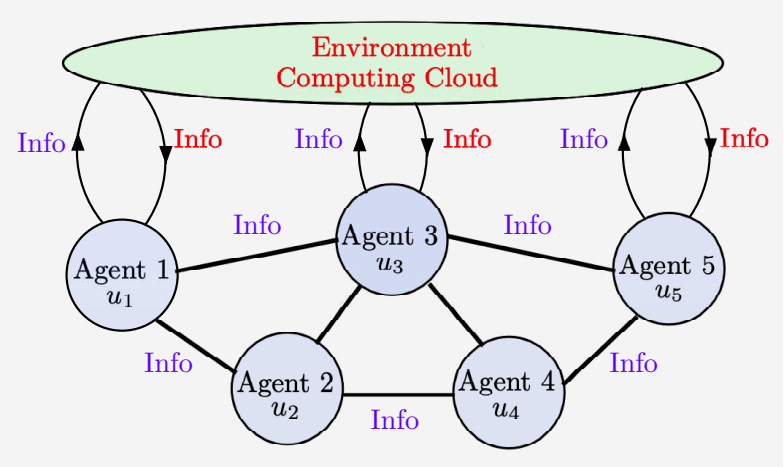


Figure 1.4.5 Schematic illustration of a multiagent problem. There are multiple "agents," and each agent $\ell = 1, \ldots, m$, controls its own decision variable u_ℓ. At each stage, agents exchange new information and also exchange information with the "environment," and then select their decision variables for the stage.

ture of the control space, whereby the control u consists of m components, $u = (u_1, \ldots, u_m)$, with a separable control constraint structure $u_\ell \in U_\ell(x)$, $\ell = 1, \ldots, m$. Thus the control constraint set is the Cartesian product

$$U(x) = U_1(x) \times \cdots \times U_m(x), \tag{1.57}$$

where the sets $U_\ell(x)$ are given. This structure is inspired by applications involving distributed decision making by multiple agents with communication and coordination between the agents; see Fig. 1.4.5.

In particular, we will view each component u_ℓ, $\ell = 1, \ldots, m$, as being chosen from within $U_\ell(x)$ by a separate "agent" (a decision making entity). For the sake of the following discussion, we assume that each set $U_\ell(x)$ is finite. Then the one-step lookahead minimization of the standard rollout scheme with base policy μ is given by

$$\tilde{u} \in \arg\min_{u \in U(x)} E_w\Big\{g(x,u,w) + \alpha J_\mu\big(f(x,u,w)\big)\Big\}, \tag{1.58}$$

and involves as many as n^m Q-factors, where n is the maximum number of elements of the sets $U_\ell(x)$ [so that n^m is an upper bound to the number of controls in $U(x)$, in view of its Cartesian product structure (1.57)]. Thus the standard rollout algorithm requires an exponential [order $O(n^m)$] number of Q-factor computations per stage, which can be overwhelming even for moderate values of m.

This potentially large computational overhead motivates a far more computationally efficient rollout algorithm, whereby the one-step lookahead

minimization (1.58) is replaced by a sequence of m successive minimizations, *one-agent-at-a-time*, with the results incorporated into the subsequent minimizations. In particular, at state x we perform the sequence of minimizations

$$\tilde{\mu}_1(x) \in \arg\min_{u_1 \in U_1(x)} E_w\Big\{g(x,u_1,\mu_2(x),\ldots,\mu_m(x),w) + \alpha J_\mu\big(f(x,u_1,\mu_2(x),\ldots,\mu_m(x),w)\big)\Big\},$$

$$\tilde{\mu}_2(x) \in \arg\min_{u_2 \in U_2(x)} E_w\Big\{g(x,\tilde{\mu}_1(x),u_2,\mu_3(x)\ldots,\mu_m(x),w) + \alpha J_\mu\big(f(x,\tilde{\mu}_1(x),u_2,\mu_3(x),\ldots,\mu_m(x),w)\big)\Big\},$$

$$\ldots \qquad \ldots \qquad \ldots \qquad \ldots$$

$$\tilde{\mu}_m(x) \in \arg\min_{u_m \in U_m(x)} E_w\Big\{g(x,\tilde{\mu}_1(x),\tilde{\mu}_2(x),\ldots,\tilde{\mu}_{m-1}(x),u_m,w) + \alpha J_\mu\big(f(x,\tilde{\mu}_1(x),\tilde{\mu}_2(x),\ldots,\tilde{\mu}_{m-1}(x),u_m,w)\big)\Big\}.$$

Thus each agent component u_ℓ is obtained by a minimization with the preceding agent components $u_1,\ldots,u_{\ell-1}$ fixed at the previously computed values of the rollout policy, and the following agent components $u_{\ell+1},\ldots,u_m$ fixed at the values given by the base policy. This algorithm requires order $O(nm)$ Q-factor computations per stage, a potentially huge computational saving over the order $O(n^m)$ computations required by standard rollout.

A key idea here is that the computational requirements of the rollout one-step minimization (1.58) are proportional to the number of controls in the set $U_k(x_k)$ and are independent of the size of the state space. This motivates a reformulation of the problem, first suggested in the book [BeT96], Section 6.1.4, whereby control space complexity is traded off with state space complexity, by "unfolding" the control u_k into its m components, which are applied *one agent-at-a-time* rather than all-agents-at-once.

In particular, we can reformulate the problem by breaking down the collective decision u_k into m sequential component decisions, thereby reducing the complexity of the control space while increasing the complexity of the state space. The potential advantage is that the extra state space complexity does not affect the computational requirements of some RL algorithms, including rollout.

To this end, we introduce a modified but equivalent problem, involving one-at-a-time agent control selection. At the generic state x, we break down the control u into the sequence of the m controls $u_1,u_2,\ldots,u_m$, and between x and the next state $\bar{x} = f(x,u,w)$, we introduce artificial intermediate "states" $(x,u_1),(x,u_1,u_2),\ldots,(x,u_1,\ldots,u_{m-1})$, and corresponding transitions. The choice of the last control component u_m at "state"

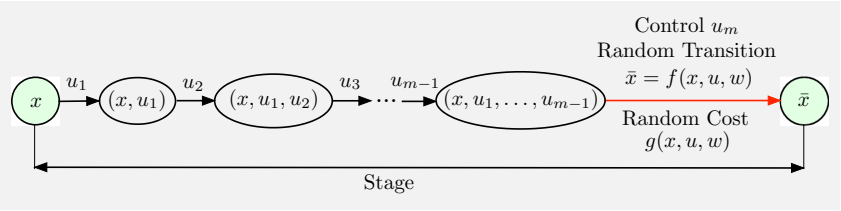


Figure 1.4.6 Equivalent formulation of the N-stage stochastic optimal control problem for the case where the control u consists of m components $u_1, u_2, \ldots, u_m$:

$$u = (u_1, \ldots, u_m) \in U_1(x_k) \times \cdots \times U_m(x_k).$$

The figure depicts the kth stage transitions. Starting from state x, we generate the intermediate states

$$(x, u_1), (x_k, u_1, u_2), \ldots, (x, u_1, \ldots, u_{m-1}),$$

using the respective controls $u_1, \ldots, u_{m-1}$. The final control u_m leads from $(x, u_1, \ldots, u_{m-1})$ to $\bar{x} = f(x, u, w)$, and the random cost $g(x, u, w)$ is incurred.

$(x, u_1, \ldots, u_{m-1})$ marks the transition to the next state $\bar{x} = f(x, u, w)$ according to the system equation, while incurring cost $g(x, u, w)$; see Fig. 1.4.6.

It is evident that this reformulated problem is equivalent to the original, since any control choice that is possible in one problem is also possible in the other problem, while the cost structure of the two problems is the same. In particular, every policy $\big(\mu_1(x), \ldots, \mu_m(x)\big)$ of the original problem, is admissible for the reformulated problem, and has the same cost function for the original as well as the reformulated problem. Reversely, every policy for the reformulated problem can be converted into a policy for the original problem that produces the same state and control trajectories and has the same cost function.

The motivation for the reformulated problem is that the control space is simplified at the expense of introducing $m-1$ additional layers of states, and the corresponding $m-1$ cost-to-go functions

$$J^1(x, u_1), J^2(x, u_1, u_2), \ldots, J^{m-1}(x, u_1, \ldots, u_{m-1}).$$

The increase in size of the state space does not adversely affect the operation of rollout, since the Q-factor minimization (1.58) is performed for just one state at each stage.

The major fact that follows from the preceding reformulation is that multiagent rollout still *achieves cost improvement*:

$$J_{\tilde{\mu}}(x) \leq J_\mu(x), \qquad \text{for all } x,$$

where $J_\mu(x)$ is the cost function of the base policy μ, and $J_{\tilde{\mu}}(x)$ is the cost function of the rollout policy $\tilde{\mu} = (\tilde{\mu}_1, \ldots, \tilde{\mu}_m)$, starting from state

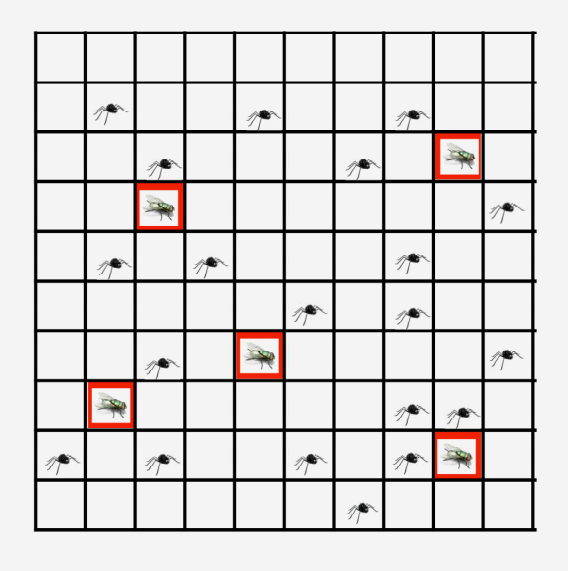


Figure 1.4.7 Illustration of a 2-dimensional spiders-and-fly problem with 20 spiders and 5 flies (cf. Example 1.4.4). The flies moves randomly, regardless of the position of the spiders. During a stage, each spider moves to a neighboring location or stays where it is, so there are 5 moves per spider (except for spiders at the edges of the grid). The total number of possible joint spider moves is a little less than 5^{20}.

x. Furthermore, this cost improvement property can be extended to multiagent PI schemes that involve one-agent-at-a-time policy improvement operations, and have sound convergence properties (see the discussion of Chapters 3 and 5). Moreover, multiagent rollout will become the starting point for various related PI schemes that are well suited for distributed operation in important practical contexts involving multiple autonomous decision makers.

The following two-dimensional version of the spider-and-fly Example 1.3.4 illustrates the multiagent rollout scheme; see Fig. 1.4.7.

Example 1.4.4 (Spiders and Flies)

This example is representative of a broad range of practical problems such as multirobot service systems involving delivery, maintenance and repair, search and rescue, firefighting, etc. Here there are m spiders and several flies moving on a 2-dimensional grid; cf. Fig. 1.4.7. The objective is for the spiders to catch all the flies as fast as possible.

During a stage, each fly moves according to a given probability distribution. Each spider learns the current state (the vector of spiders and fly locations) at the beginning of each stage, and either moves to a neighboring location or stays where it is. Thus each spider has as many as 5 choices at

each stage. The control is $u = (u_1, \ldots, u_m)$, where u_ℓ is the choice of the ℓth spider, so there are about 5^m possible values of u.

To apply multiagent rollout, we need a base policy. A simple possibility is to use the policy that directs each spider to move on the path of minimum distance to the closest fly position. According to the multiagent rollout formalism, the spiders choose their moves one-at-time in the order $1, \ldots, m$, taking into account the current positions of the flies and the earlier moves of other spiders, while assuming that future moves will be chosen according to the base policy. This is a tractable computation.

In particular, at the beginning at the typical stage, spider 1 selects its best move (out of the no more than 5 possible moves), assuming the other spiders $2, \ldots, m$ will move towards their closest surviving fly during the current stage, and all spiders will move towards their closest surviving fly during the following stages, up to the time where no surviving flies remain. Spider 1 then broadcasts its selected move to all other spiders. Then spider 2 selects its move taking into account the move already chosen by spider 1, and assuming that spiders $3, \ldots, m$ will move towards their closest surviving fly during the current stage, and all spiders will move towards their closest surviving fly during the following stages, up to the time where no surviving flies remain. Spider 2 then broadcasts its choice to all other spiders. This process of one-spider-at-a-time move selection is repeated for the remaining spiders $3, \ldots, m$, marking the end of the stage.

Note that while standard rollout computes and compares 5^m Q-factors (actually a little less to take into account edge effects), multiagent rollout computes and compares ≤ 5 moves per spider, for a total of less than $5m$. Despite this tremendous computational economy, experiments with this type of spiders and flies problems have shown that multiagent rollout achieves a comparable performance to the one of standard rollout.

1.4.6 Problems with Unknown Parameters - Adaptive Control

Our discussion so far dealt with problems with a known mathematical model, i.e., one where the system equation, cost function, control constraints, and probability distributions of disturbances are perfectly known. The mathematical model may be available through explicit mathematical formulas and assumptions, or through a computer program that can emulate all of the mathematical operations involved in the model, including Monte Carlo simulation for the calculation of expected values.

In this connection, it is important to note that from our point of view, *it makes no difference whether the mathematical model is available through closed form mathematical expressions or through a computer simulator*: the methods that we discuss are valid either way, only their suitability for a given problem may be affected by the availability of mathematical formulas. Indeed, problems with a known mathematical model are the only type that we will formally address in this book with DP and approximate DP methods. In particular, we will not discuss model estimation methods, except peripherally, as in the present section.

In practice, however, it is common that the system parameters are either not known exactly or may change over time. In such cases it is important to design controllers that take the parameter changes into account. The methodology for doing so is generally known as *adaptive control*. This is an intricate and multifaceted methodology, with many and diverse applications, and a long history.†

As an example consider our oversimplified cruise control system of Example 1.3.1 or its infinite horizon version of Examples 1.3.3 and 1.3.5. The state evolves according to

$$x_{k+1} = x_k + bu_k + w_k, \tag{1.59}$$

where x_k is the deviation $v_k - \bar{v}$ of the vehicle's velocity v_k from the nominal $\bar{v}$, u_k is the force that propels the car forward, and w_k is the disturbance that has nonzero mean. However, the coefficient b and the distribution of w_k change frequently, and cannot be modeled with any precision because they depend on unpredictable time-varying conditions, such as the slope and condition of the road, and the weight of the car (which is affected by the number of passengers). Moreover, the nominal velocity $\bar{v}$ is set by the driver, and when it changes it may affect the parameter b in the system equation, and other parameters.‡

We should note also that unknown problem environments are an integral part of the artificial intelligence view of RL. In particular, to quote from the book by Sutton and Barto [SuB18], "learning from interaction with the environment is a foundational idea underlying nearly all theories of learning and intelligence." The idea of interaction with the environment is typically connected with the idea of exploring the environment to identify its characteristics. In control theory this is often viewed as part of the *system identification* methodology, which aims to construct mathematical models of dynamic systems. The system identification process is often combined with the control process to deal with unknown or changing problem parameters. This is one of the most challenging areas of stochastic optimal and suboptimal control, and has been studied since the early 1960s.

In this book we will not provide a systematic discussion of adaptive control. Instead, in what follows in this section, we will briefly review some

† The difficulties of designing adaptive controllers are often underestimated. Among others, they complicate the balance between off-line training and on-line play, which we discussed in Section 1.1 in connection to AlphaZero. It is worth keeping in mind that as much as learning to play high quality chess is a great challenge, the rules of play are stable and do not change unpredictably in the middle of a game! Problems with changing system parameters can be far more challenging!

‡ Adaptive cruise control, which can also adapt the car's velocity based on its proximity to other cars, has been studied extensively and has been incorporated in several commercially sold car models.

of the principal types of adaptive control methods. We will then focus on schemes that are based on on-line replanning, including the use of rollout, which we will discuss in greater detail in the next section.

Robust and PID Control

Given a controller design that has been obtained assuming a nominal DP problem model, one possibility is to simply ignore changes in problem parameters. We may then try to design a controller that is adequate for the entire range of the changing parameters. This is sometimes called a *robust controller*. For example, consider the oversimplified cruise control system of Eq. (1.59), where we choose a linear controller of the form $\mu(x) = Lx$ for some scalar L. Then we may perform a robustness analysis, i.e., determine the range of parameters b for which the current controller is stable (this is the interval of values b for which $|1 + bL| < 1$), and check that b remains within that range during the system's operation. Thus, a robust controller makes no effort to keep track of changing problem parameters. It is just designed so that it is resilient to parameter changes.

An important time-honored robust control approach for continuous-state problems is the *PID (Proportional-Integral-Derivative) controller*; see e.g., the books by Aström and Hagglund [AsH95], [AsH06], and the end-of-chapter references. In particular, PID control aims to maintain the output of a single-input single-output dynamic system around a set point or to follow a given trajectory, as the system parameters change within a relatively broad range. In its simplest form, the PID controller is parametrized by three scalar parameters, which may be determined by a variety of methods, some of them manual/heuristic. PID control is used widely and with success, although its range of application is mainly restricted to single-input, single-output continuous-state control systems.

Dealing with Unknown Parameters by System Identification and On-Line Replanning

In robust control schemes, such as PID control, no attempt is made to maintain a mathematical model and to track unknown model parameters as they change. Alternatively we may introduce into the controller a mechanism for measuring or estimating the unknown or changing system parameters, and make suitable control adaptations in response.†

† In the adaptive control literature, schemes that involve parameter estimation are sometimes called *indirect*, while schemes that do not involve parameter estimation (like PID control) are called *direct*. To quote from the book by Aström and Wittenmark [AsW08], "indirect methods are those in which the estimated parameters are used to calculate required controller parameters" (see Fig. 1.4.8). The methods subsequently described in this section, and the rollout-based adaptive control methods discussed in the next section should be viewed as indirect.

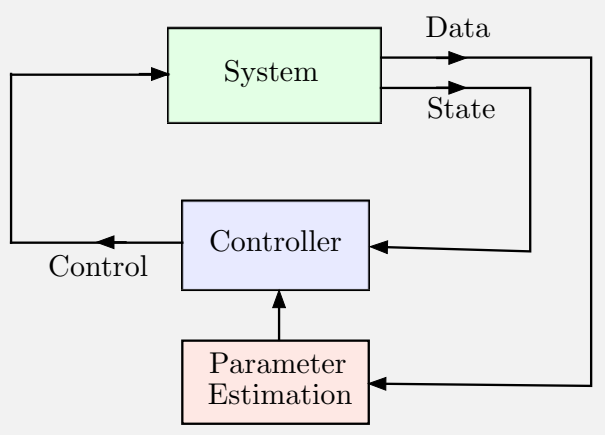


Figure 1.4.8 Schematic illustration of concurrent parameter estimation and system control. The system parameters are estimated on-line and the estimates are passed on to the controller whenever this is desirable (e.g., after the estimates change substantially). This structure is also known as indirect adaptive control.

An apparently reasonable scheme is to separate the control process into two phases, a *system identification phase* and a *control phase*. In the first phase the unknown parameters are estimated, while the control takes no account of the interim results of estimation. The final parameter estimates from the first phase are then used to implement an optimal or suboptimal policy in the second phase.

This alternation of estimation and control phases may be repeated several times during the system's operation in order to take into account subsequent changes of the parameters. Note that it is not necessary to introduce a hard separation between the identification and the control phases. They may be going on simultaneously, with new parameter estimates being generated in the background, and introduced into the control process, whenever this is thought to be desirable; see Fig. 1.4.8.

One drawback of this approach is that it is not always easy to determine when to terminate one phase and start the other. A second difficulty, of a more fundamental nature, is that the control process may make some of the unknown parameters invisible to the estimation process. This is known as the problem of *parameter identifiability*, which is discussed in the context of optimal control in several sources, including [BoV79] and [Kum83]; see also [Ber17a], Section 6.7. Here is a simple illustrative example.

Example 1.4.5 (On-Line Identification in Linear Quadratic Optimal Control)

Consider the scalar system

$$x_{k+1} = ax_k + bu_k, \qquad k = 0, \ldots, N-1,$$

and the quadratic cost

$$\sum_{k=1}^{N}(x_k)^2.$$

Assuming perfect state information, if the parameters a and b are known, it can be seen that the optimal policy is

$$\mu_k^*(x_k) = -\frac{a}{b}x_k,$$

which sets all future states to 0. Assume now that the parameters a and b are unknown, and consider the two-phase method. During the first phase the policy

$$\tilde{\mu}_k(x_k) = \gamma x_k \tag{1.60}$$

is used (γ is some scalar; for example, $\gamma = -\overline{a}/\overline{b}$, where $\overline{a}$ and $\overline{b}$ are some a priori estimates of a and b, respectively). At the end of the first phase, the policy is changed to

$$\overline{\mu}_k(x_k) = -\frac{\hat{a}}{\hat{b}}x_k,$$

where $\hat{a}$ and $\hat{b}$ are the estimates obtained from the estimation process. However, with the policy (1.60), the closed-loop system is

$$x_{k+1} = (a + b\gamma)x_k,$$

so the estimation process can at best yield the value of $(a + b\gamma)$ but not the values of both a and b. In other words, the estimation process cannot discriminate between pairs of values (a_1, b_1) and (a_2, b_2) such that $a_1 + b_1\gamma = a_2 + b_2\gamma$. Therefore, a and b are not identifiable when feedback control of the form (1.60) is applied.

On-line parameter estimation algorithms, which address among others the issue of identifiability, have been discussed extensively in the control theory literature, but the corresponding methodology is complex and beyond our scope in this book. However, assuming that we can make the estimation phase work somehow, we are free to reoptimize the controller using the newly estimated parameters, in a form of on-line replanning process.

Unfortunately, there is still another difficulty with this type of on-line replanning: it may be hard to recompute an optimal or near-optimal policy on-line, using a newly identified system model. In particular, it may be impossible to use time-consuming methods that involve for example the training of a neural network, or discrete/integer control constraints.† A simpler possibility is to use rollout, which we discuss in the next section.

† Another possibility is to deal with this difficulty by precomputation. In particular, assume that the set of problem parameters may take a known finite set of values (or example each set of parameter values may correspond to a distinct

1.4.7 Adaptive Control by Rollout and On-Line Replanning

We will now consider an approach for dealing with unknown or changing parameters, which is based on rollout and on-line replanning. We have already noted this approach in Sections 1.1 and 1.2, where we stressed the importance of fast on-line policy improvement.

Let us assume that some problem parameters change over time and the controller becomes aware of the changes, perhaps after a suitable delay for data collection and estimation. The method by which the problem parameters are recalculated or become known is immaterial for the purposes of the following discussion. It may involve a limited form of parameter estimation, whereby the unknown parameters are "tracked" by data collection over a few time stages, with due attention paid to issues of parameter identifiability; or it may involve new features of the control environment, such as a changing number of servers and/or tasks in a service system (think of new spiders and/or flies appearing or disappearing unexpectedly in the spiders-and-flies Example 1.4.4).

We thus assume away/ignore the detailed issues of parameter estimation, and focus on revising the controller by on-line replanning based on the newly obtained parameters. This revision may be based on any suboptimal method, but rollout with some base policy is particularly attractive. The base policy may be either a fixed robust controller (such as some form of PID control) or it may be updated over time (in the background, on the basis of some unspecified rationale), in which case the rollout policy will be revised both in response to the changed base policy and in response to the changing parameters.

Here the advantage of rollout is that it is simple, reliable, and relatively fast. In particular, it does not require a complicated training procedure, based for example on the use of neural networks or other approximation architectures, so *no new policy is explicitly computed in response to the parameter changes*. Instead the available controls at the current state are compared by a one-step or multistep minimization, with cost function approximation provided by the base policy (cf. Fig. 1.4.9).

Another issue to consider is the stability and robustness properties of the rollout policy. In this connection, see Exercises 1.6 and 3.2, which suggest that *if the base policy is stable within a range of parameter values, the same is true for the rollout policy*; this can also be inferred from Fig. 1.3.12. Related ideas have a long history in the control theory literature;

---

maneuver of a vehicle, motion of a robotic arm, flying regime of an aircraft, etc). Then we may precompute a separate controller for each of these values. Once the control scheme detects a change in problem parameters, it switches to the corresponding predesigned current controller. This is sometimes called a *multiple model control design* or *gain scheduling*, and has been applied with success in various settings over the years.

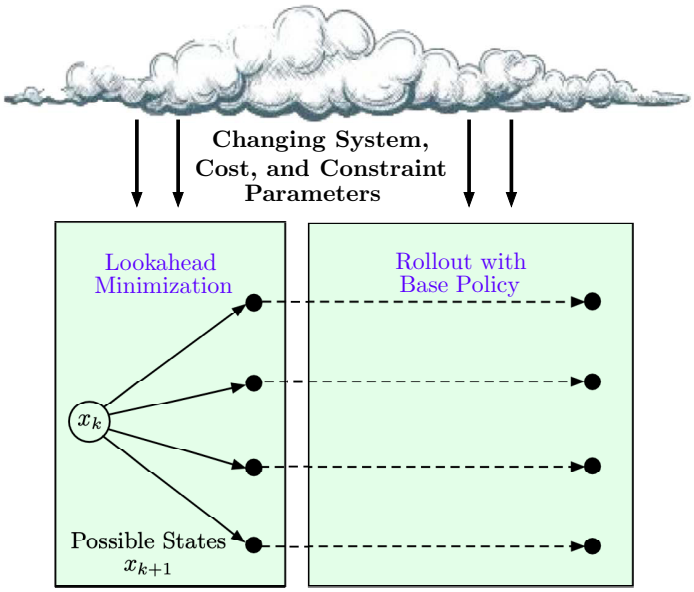


Figure 1.4.9 Schematic illustration of adaptive control by on-line replanning based on rollout. One-step lookahead is followed by simulation with the base policy, which stays fixed. The system, cost, and constraint parameters are changing over time, and the most recent estimates of their values are incorporated into the lookahead minimization and rollout operations. Truncated rollout with terminal cost approximation is also possible. For the discussion in this section, we may assume that all the changing parameter information is provided by some computation and sensor "cloud" that is beyond our control. The base policy may also be revised based on various criteria.

see Beard [Bea95], Beard, Saridis, and Wen [BSW99], Jiang and Jiang [JiJ17], Kalise, Kundu, Kunisch [KKK20].

The principal requirement for using rollout in an adaptive control context is that the rollout control computation should be fast enough to be performed between stages. In this connection, we note that accelerated/truncated or simplified versions of rollout, as well as parallel computation, can be used to meet this time constraint.

Generally, adaptive control by rollout and on-line replanning makes sense in situations where the calculation of the rollout controls for a given set of problem parameters is faster and/or more convenient than the calculation of the optimal controls for the same set of parameter values. These problems include cases involving nonlinear systems and/or difficult (e.g., integer) constraints.

The following example illustrates on-line replanning with the use of rollout in the context of the simple one-dimensional linear quadratic problem that we discussed earlier in this chapter. The purpose of the example is to show analytically how rollout with a base policy that is optimal for a

nominal set of problem parameters works well when the parameters change from their nominal values. This property is not practically useful in linear quadratic problems because when the parameter change, it is possible to calculate the new optimal policy in closed form, but it is indicative of the performance robustness of rollout in other contexts; for example linear quadratic problems with constraints.

Example 1.4.6 (On-Line Replanning for Linear Quadratic Problems Based on Rollout)

Consider the deterministic undiscounted infinite horizon linear quadratic problem of Example 1.3.3. It involves the linear system

$$x_{k+1} = x_k + bu_k,$$

and the quadratic cost function

$$\lim_{N\to\infty} \sum_{k=0}^{N-1} (x_k^2 + ru_k^2).$$

The optimal cost function is given by

$$J^*(x) = K^* x^2,$$

where K^* is the unique positive solution of the Riccati equation

$$K = \frac{rK}{r + b^2 K} + 1. \tag{1.61}$$

The optimal policy has the form

$$\mu^*(x) = L^* x, \tag{1.62}$$

where

$$L^* = -\frac{bK^*}{r + b^2 K^*}. \tag{1.63}$$

As an example, consider the optimal policy that corresponds to the nominal problem parameters $b = 2$ and $r = 0.5$: this is the policy (1.62)-(1.63), with K obtained as the positive solution of the quadratic Riccati Eq. (1.61) for $b = 2$ and $r = 0.5$. We thus obtain

$$K = \frac{2 + \sqrt{6}}{4}.$$

From Eq. (1.63) we then have

$$L = -\frac{2 + \sqrt{6}}{5 + 2\sqrt{6}}. \tag{1.64}$$

We will now consider changes of the values of b and r while keeping L constant, and we will compare the quadratic cost coefficient of the following three cost functions as b and r vary:

(a) The optimal cost function K^*x^2, where K^* is given by the positive solution of the Riccati Eq. (1.61).

(b) The cost function K_Lx^2 that corresponds to the base policy

$$\mu_L(x) = Lx,$$

where L is given by Eq. (1.64). Here, from Example 1.3.5, we have

$$K_L = \frac{1 + rL^2}{1 - (1 + bL)^2}. \tag{1.65}$$

(c) The cost function $\tilde{K}_Lx^2$ that corresponds to the rollout policy

$$\tilde{\mu}_L(x) = \tilde{L}x,$$

obtained by using the policy μ_L as base policy. Using the formulas of Example 1.3.5, we have

$$\tilde{L} = -\frac{bK_L}{r + b^2K_L}, \tag{1.66}$$

and

$$\tilde{K}_L = \frac{1 + r\tilde{L}^2}{1 - (1 + b\tilde{L})^2}.$$

Figure 1.4.10 shows the coefficients K^*, K_L, and $\tilde{K}_L$ for a range of values of r and b. We have

$$K^* \leq \tilde{K}_L \leq K_L.$$

The difference $K_L - K^*$ is indicative of the robustness of the policy μ_L, i.e., the performance loss incurred by ignoring the values of b and r, and continuing to use the policy μ_L, which is optimal for the nominal values $b = 2$ and $r = 0.5$, but suboptimal for other values of b and r. The difference $\tilde{K}_L - K^*$ is indicative of the performance loss due to using on-line replanning by rollout rather than using optimal replanning. Finally, the difference $K_L - \tilde{K}_L$ is indicative of the performance improvement due to on-line replanning using rollout rather than keeping the policy μ_L unchanged.

Note that Fig. 1.4.10 illustrates the behavior of the error ratio

$$\frac{\tilde{J} - J^*}{J - J^*},$$

where for a given initial state, $\tilde{J}$ is the rollout performance, J^* is the optimal performance, and J is the base policy performance. This ratio approaches 0 as $J - J^*$ becomes smaller because of the superlinear/quadratic convergence rate of Newton's method that underlies the rollout algorithm (cf. Section 1.3.4).

The next example summarizes how rollout and on-line replanning relate to model predictive control (MPC). A detailed discussion will be given in Chapter 3; see also the author's papers [Ber05a], [Ber05b], where the relations between rollout and MPC were first explored.

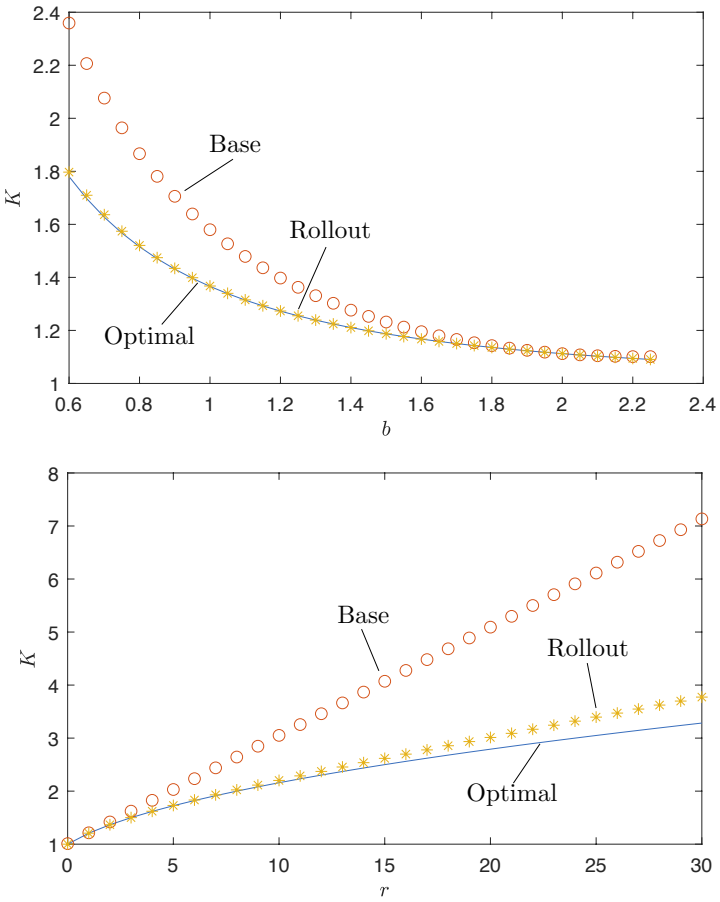


Figure 1.4.10 Illustration of control by rollout under changing problem parameters. The quadratic cost coefficients K^* (optimal, denoted by solid line), K_L (base policy, denoted by circles), and $\tilde{K}_L$ (rollout policy, denoted by asterisks) for the two cases where $r = 0.5$ and b varies, and $b = 2$ and r varies. The value of L is fixed at the value that is optimal for $b = 2$ and $r = 0.5$ [cf. Eq. (1.64)]. The rollout policy performance is very close to optimal, even when the base policy is far from optimal.

Note that, as the figure illustrates, we have

$$\lim_{J \to J^*} \frac{\tilde{J} - J^*}{J - J^*} = 0,$$

where for a given initial state, $\tilde{J}$ is the rollout performance, J^* is the optimal performance, and J is the base policy performance. This is a consequence of the superlinear/quadratic convergence rate of Newton's method that underlies rollout, and guarantees that the rollout performance approaches the optimal much faster than the base policy performance does (cf. Section 1.3.4).

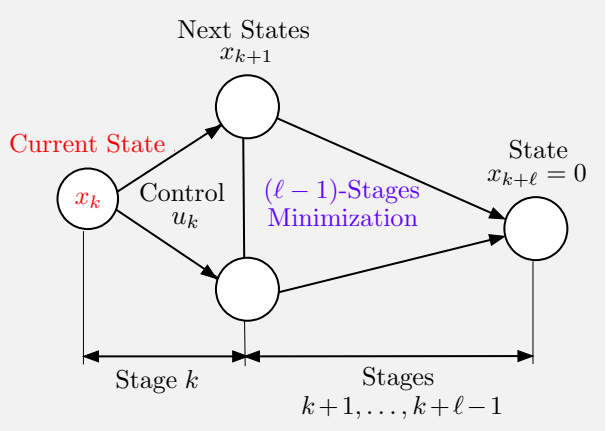


Figure 1.4.11 Illustration of the problem solved by MPC at state x_k. We minimize the cost function over the next ℓ stages while imposing the requirement that $x_{k+\ell}=0$. We then apply the first control of the optimizing sequence. In the context of rollout, the minimization over u_k is the one-step lookahead, while the minimization over $u_{k+1},\ldots,u_{k+\ell-1}$ that drives $x_{k+\ell}$ to 0 is the base heuristic.

Example 1.4.7 (Model Predictive Control, Rollout, and On-Line Replanning)

Let us briefly discuss the MPC methodology, with a view towards its connection with the rollout algorithm. Consider an undiscounted infinite horizon deterministic problem, involving the system

$$x_{k+1}=f(x_k,u_k),$$

whose state x_k and control u_k are finite-dimensional vectors. The cost per stage is assumed nonnegative

$$g(x_k,u_k)\geq 0, \qquad \text{for all } (x_k,u_k),$$

(e.g., a positive definite quadratic cost). There are control constraints $u_k\in U(x_k)$, and to simplify the following discussion, we will assume that there are no state constraints. We assume that the system can be kept at the origin at zero cost, i.e.,

$$f(0,\overline{u}_k)=0, \quad g(0,\overline{u}_k)=0 \qquad \text{for some control } \overline{u}_k\in U(0).$$

For a given initial state x_0, we want to obtain a sequence $\{u_0,u_1,\ldots\}$ that satisfies the control constraints, while minimizing the total cost. This is a classical problem in control system design, where the aim is to keep the state of the system near the origin (or more generally some desired set point), in the face of disturbances and/or parameter changes.

The MPC algorithm at each encountered state x_k applies a control that is computed as follows; see Fig. 1.4.11:

(a) It solves an ℓ-stage optimal control problem involving the same cost function and the requirement $x_{k+\ell} = 0$. This is the problem

$$\min_{u_t,\, t=k,\ldots,k+\ell-1} \sum_{t=k}^{k+\ell-1} g(x_t, u_t), \tag{1.67}$$

subject to the system equation constraints

$$x_{t+1} = f(x_t, u_t), \qquad t = k, \ldots, k+\ell-1,$$

the control constraints

$$u_t \in U(x_t), \qquad t = k, \ldots, k+\ell-1,$$

and the terminal state constraint $x_{k+\ell} = 0$. Here ℓ is an integer with $\ell > 1$, which is chosen in some largely empirical way (see Section 3.1 for further discussion).

(b) If $\{\tilde{u}_k, \ldots, \tilde{u}_{k+\ell-1}\}$ is the optimal control sequence of this problem, MPC applies $\tilde{u}_k$ and discards the other controls $\tilde{u}_{k+1}, \ldots, \tilde{u}_{k+\ell-1}$.

(c) At the next stage, MPC repeats this process, once the next state x_{k+1} is revealed.

To make the connection of MPC with rollout, we note that *the one-step lookahead function $\tilde{J}$ implicitly used by MPC [cf. Eq. (1.67)] is the cost-to-go function of a certain base heuristic*. This is the heuristic that drives to 0 the state after $\ell - 1$ stages (*not ℓ stages*) and keeps the state at 0 thereafter, while observing the state and control constraints, and minimizing the associated $(\ell - 1)$-stages cost. This rollout view of MPC, first discussed in the author's paper [Ber05a], is useful for making a connection with the approximate DP/RL techniques that are the main subject of this book, such as truncated rollout, cost function approximations, PI, etc.

Returning to the issue of dealing with changing problem parameters, it is natural to consider on-line replanning as per our earlier discussion. In particular, once new estimates of system and/or cost function parameters become available, MPC can adapt accordingly by introducing the new parameter estimates into the ℓ-stage optimization problem in (a) above.

Let us also note a common variant of MPC, where the requirement of driving the system state to 0 in ℓ steps in the ℓ-stage MPC problem (1.67), is replaced by a terminal cost $G(x_{k+\ell})$ (other variants, which include state constraints, will be discussed in Section 3.1). Thus at state x_k, we solve the problem

$$\min_{u_t,\, t=k,\ldots,k+\ell-1} \left[G(x_{k+\ell}) + \sum_{t=k}^{k+\ell-1} g(x_t, u_t) \right],$$

instead of problem (1.67) where we require that $x_{k+\ell} = 0$. This variant can also be viewed as rollout with one-step lookahead, and a base heuristic, which

at state x_{k+1} applies the first control $\tilde{u}_{k+1}$ of the sequence $\{\tilde{u}_{k+1}, \ldots, \tilde{u}_{k+\ell-1}\}$ that minimizes

$$G(x_{k+\ell}) + \sum_{t=k+1}^{k+\ell-1} g(x_t, u_t).$$

Note that the preceding MPC controller may outperform substantially its base heuristic (in relative terms), particularly if the base heuristic is close to optimal [which is true if $G(x_{k+\ell}) \approx J^*(x_{k+\ell})+$ a constant]. This is because in view of the superlinear/quadratic convergence rate of Newton's method that underlies rollout, we have

$$\lim_{J \to J^*} \frac{\tilde{J} - J^*}{J - J^*} = 0,$$

where for a given initial state, $\tilde{J}$ is the rollout performance, J^* is the optimal performance, and J is the base policy performance (cf. Section 1.3.4, Fig. 1.4.10, and the discussion in Section 3.1).

1.5 REINFORCEMENT LEARNING AND OPTIMAL CONTROL - SOME TERMINOLOGY

The current state of RL has greatly benefited from the cross-fertilization of ideas from optimal control and from artificial intelligence. The strong connections between these two fields are now widely recognized. Still, however, substantial differences in language and emphasis remain between RL-based discussions (where artificial intelligence-related terminology is used) and DP-based discussions (where optimal control-related terminology is used).

The terminology used in this book is standard in DP and optimal control, and in an effort to forestall confusion of readers that are accustomed to either the artificial intelligence or the optimal control terminology, we provide a list of terms commonly used in RL, and their optimal control counterparts.

(a) **Environment** = System.

(b) **Agent** = Decision maker or controller.

(c) **Action** = Decision or control.

(d) **Reward of a stage** = (Opposite of) Cost of a stage.

(e) **State value** = (Opposite of) Cost starting from a state.

(f) **Value (or reward) function** = (Opposite of) Cost function.

(g) **Maximizing the value function** = Minimizing the cost function.

(h) **Action (or state-action) value** = Q-factor (or Q-value) of a state-control pair. (Q-value is also used often in RL.)

(i) **Planning** = Solving a DP problem with a known mathematical model.

(j) **Learning** = Solving a DP problem without using an explicit mathematical model. (This is the principal meaning of the term "learning" in RL. Other meanings are also common.)

(k) **Self-learning** (or self-play in the context of games) = Solving a DP problem using some form of policy iteration.

(l) **Deep reinforcement learning** = Approximate DP using value and/or policy approximation with deep neural networks.

(m) **Prediction** = Policy evaluation.

(n) **Generalized policy iteration** = Optimistic policy iteration.

(o) **State abstraction** = State aggregation.

(p) **Temporal abstraction** = Time aggregation.

(q) **Learning a model** = System identification.

(r) **Episodic task or episode** = Finite-step system trajectory.

(s) **Continuing task** = Infinite-step system trajectory.

(t) **Experience replay** = Reuse of samples in a simulation process.

(u) **Bellman operator** = DP mapping or operator (the name "Bellman operator" is also used in the optimal control literature).

(v) **Backup** = Applying the DP operator at some state.

(w) **Greedy policy with respect to a cost function** J = Minimizing policy in the DP expression defined by J.

(x) **Afterstate** = Post-decision state.

(y) **Generative model** = Computer simulator of a controlled system.

(z) **Ground truth** = Empirical evidence or information provided by direct observation.

Some of the preceding terms will be introduced in future chapters; see also the RL textbook [Ber19a]. The reader may then wish to return to this section as an aid in connecting with the relevant RL literature.

Notation

Unfortunately the confusion arising from different terminology has been exacerbated by the use of different notations. The present book roughly follows the "standard" notation of the Bellman/Pontryagin optimal control era; see e.g., the classical books by Athans and Falb [AtF66], Bellman [Bel67], and Bryson and Ho [BrH75]. This notation is the most appropriate for a unified treatment of the subject, which simultaneously addresses discrete and continuous spaces problems, as well as deterministic problems for which the MDP-style notation is unsuitable.

A summary of our most prominently used symbols is as follows:

(a) x: state.

(b) u: control.

(c) J: cost function.

(d) g: cost per stage.

(e) f: system function.

(f) i: discrete state.

(g) $p_{xy}(u)$: transition probability from state x to state y under control u.

(h) α: discount factor in discounted problems.

The x-u-J notation is standard in optimal control textbooks (e.g., the books by Athans and Falb [AtF66], and Bryson and Ho [BrH75], as well as the more recent book by Liberzon [Lib11]). The notations f and g are also used most commonly in the literature of the early optimal control period as well as later (unfortunately the more natural symbol "c" has not been used much in place of "g" for the cost per stage). The discrete system notations i and $p_{ij}(u)$ are very common in the operations research literature, where discrete-state MDP have been treated extensively [sometimes the alternative notation $p(j \mid i, u)$ is used for the transition probabilities].

The artificial intelligence literature addresses for the most part finite-state Markov decision problems, most frequently the discounted and stochastic shortest path infinite horizon problems that are discussed in Chapter 5. The most commonly used notation is s for state, a for action, $r(s, a, s')$ for reward per stage, $p(s' \mid s, a)$ or $p(s, a, s')$ for transition probability from s to s' under action a, and γ for discount factor. However, this type of notation is not well suited for continuous spaces models, which are of major interest for this book. The reason is that it requires the use of transition probability distributions defined over continuous spaces, and it leads to more complex and less intuitive mathematics. Moreover the transition probability notation makes no sense for deterministic problems, which involve no probabilistic structure at all.

1.6 NOTES AND SOURCES

In this section we first provide an overview of the DP and RL literature that supplements our presentation, focusing primarily on books and research monographs. We then summarize our rollout and policy iteration (PI) focus, how it relates to the experiences with AlphaZero and TD-Gammon, and what the implications are for decision and control problems.

Exact DP and Approximate DP/RL Books

Our discussion of exact DP in this chapter has been brief since our focus in this book will be on approximate DP and RL. The author's DP textbook

[Ber17a] provides an extensive discussion of finite horizon exact DP, and its applications to discrete and continuous spaces problems, using a notation and style that is consistent with the one used here. The books by Puterman [Put94] and by the author [Ber12] provide detailed treatments of infinite horizon finite-state stochastic DP problems, including average cost infinite horizon problems, which we will not consider at all here. The book [Ber12] also covers continuous/infinite state and control spaces problems, including the linear quadratic problems that we have discussed in this chapter through examples. Continuous spaces problems present special analytical and computational challenges, which are currently at the forefront of RL research.

Some of the more complex mathematical aspects of exact DP are discussed in the monograph by Bertsekas and Shreve [BeS78], particularly the probabilistic/measure-theoretic issues associated with stochastic optimal control, including partial state information problems, by using universally measurable policies. The book [Ber12] provides in an appendix an accessible summary introduction of the measure-theoretic framework of the book [BeS78], while a long paper by Yu and Bertsekas [YuB15] contains recent supplementary research with a particular focus on the analysis of PI methods, which were not treated in [BeS78]. The papers by Yu [Yu20], [Yu21] consider average cost infinite horizon DP problems with universally measurable policies.

The author's abstract DP monograph [Ber18a] aims at a unified development of the core theory and algorithms of total cost sequential decision problems, and addresses simultaneously stochastic, minimax, game, risk-sensitive, and other DP problems, through the use of abstract DP operators [or Bellman operators as they are often called in RL; Eqs. (1.38) and (1.39) provide the Bellman operators for discounted infinite horizon problems]. The idea here is to gain insight through abstraction. In particular, the structure of a DP model is encoded in its abstract Bellman operator, which serves as the "mathematical signature" of the model. Thus, characteristics of this operator (such as monotonicity and contraction) largely determine the analytical results and computational algorithms that can be applied to that model. This abstract viewpoint has been adopted in Sections 5.5-5.8 of the present book.

The approximate DP and RL literature has expanded tremendously since the connections between DP and RL became apparent in the late 80s and early 90s. We restrict ourselves to mentioning textbooks and research monographs, which supplement our discussion, express related viewpoints, and collectively provide a guide to the literature.

Two books were written in the 1990s, setting the tone for subsequent developments in the field. One in 1996 by Bertsekas and Tsitsiklis [BeT96], which reflects a decision, control, and optimization viewpoint, and another in 1998 by Sutton and Barto, which reflects an artificial intelligence viewpoint (a 2nd edition, [SuB18], was published in 2018). We refer to the

former book and also to the author's DP textbooks [Ber12], [Ber17a] for a broader discussion of some of the topics of this book, including algorithmic convergence issues and additional DP models, such as those based on average cost and semi-Markov problem optimization. Note that both of the books [BeT96] and [SuB18] deal with finite-state Markovian decision models and use a transition probability notation, as they do not address continuous spaces problems, which are of major interest in this book.

More recent books are by Gosavi [Gos15] (a much expanded 2nd edition of his 2003 monograph), which emphasizes simulation-based optimization and RL algorithms, Cao [Cao07], which focuses on a sensitivity approach to simulation-based DP methods, Chang, Hu, Fu, and Marcus [CHF13] (a 2nd edition of their 2007 monograph), which emphasizes finite-horizon/multistep lookahead schemes and adaptive sampling, Busoniu, Babuska, De Schutter, and Ernst [BBD10a], which focuses on function approximation methods for continuous space systems and includes a discussion of random search methods, Szepesvari [Sze10], which is a short monograph that selectively treats some of the major RL algorithms such as temporal differences, armed bandit methods, and Q-learning, Powell [Pow11], which emphasizes resource allocation and operations research applications, Powell and Ryzhov [PoR12], which focuses on specialized topics in learning and Bayesian optimization, Vrabie, Vamvoudakis, and Lewis [VVL13], which discusses neural network-based methods and on-line adaptive control, Kochenderfer et al. [KAC15], which selectively discusses applications and approximations in DP and the treatment of uncertainty, Jiang and Jiang [JiJ17], which addresses adaptive control and robustness issues within an approximate DP framework, Liu, Wei, Wang, Yang, and Li [LWW17], which deals with forms of adaptive dynamic programming, and topics in both RL and optimal control, and Zoppoli, Sanguineti, Gnecco, and Parisini [ZSG20], which addresses neural network approximations in optimal control as well as multiagent/team problems with nonclassical information patterns.

There are also several books that, while not exclusively focused on DP and/or RL, touch upon several of the topics of this book. The book by Borkar [Bor08] is an advanced monograph that addresses rigorously many of the convergence issues of iterative stochastic algorithms in approximate DP, mainly using the so called ODE approach. The book by Meyn [Mey07] is broader in its coverage, but discusses some of the popular approximate DP/RL algorithms. The book by Haykin [Hay08] discusses approximate DP in the broader context of neural network-related subjects. The book by Krishnamurthy [Kri16] focuses on partial state information problems, with discussion of both exact DP, and approximate DP/RL methods. The book by Brandimarte [Bra21] is a tutorial introduction to DP/RL that emphasizes operations research applications and includes MATLAB codes.

The present book is similar in style, terminology, and notation to the author's recent RL textbook [Ber19a], which provides a more com-

prehensive account of the subject. It particular, this textbook includes a broader coverage of approximation in value space methods, including certainty equivalent control and aggregation methods. It also covers substantially policy gradient and random search methods for approximation in policy space, which we will not address here. Moreover, it provides the proofs for some of the analytical and computational infinite horizon results and error bounds, which we give here without proofs.

In addition to textbooks, there are many surveys and short research monographs relating to our subject, which are rapidly multiplying in number. We refer to the author's RL textbook [Ber19a] for a fairly comprehensive list up to 2019. We next provide some more specific comments and references that supplement the material of this chapter.

Comments and References

Sections 1.1-1.3: Approximation in value space, rollout, and policy iteration (PI) are the principal subjects of this book.† These are very powerful and general techniques: they can be applied to deterministic and stochastic problems, finite and infinite horizon problems, discrete and continuous spaces problems, and mixtures thereof. Rollout is reliable, easy to implement, and can be used in conjunction with on-line replanning.

As we have noted, rollout with a given base policy is simply the first iteration of the PI algorithm starting from the base policy. Truncated rollout will be interpreted in Chapter 5 as an "optimistic" form of PI, whereby a policy is evaluated inexactly, by using a limited number of value iterations.‡

† The name "rollout" (also called "policy rollout") was introduced by Tesauro and Galperin [TeG96] in the context of rolling the dice in the game of backgammon. In Tesauro's proposal, a given backgammon position is evaluated by "rolling out" many games starting from that position to the end of the game. To quote from the paper [TeG96]: "In backgammon parlance, the expected value of a position is known as the "equity" of the position, and estimating the equity by Monte-Carlo sampling is known as performing a "rollout." This involves playing the position out to completion many times with different random dice sequences, using a fixed policy P to make move decisions for both sides."

‡ Truncated rollout was also proposed in [TeG96]. To quote from this paper: "Using large multi-layer networks to do full rollouts is not feasible for real-time move decisions, since the large networks are at least a factor of 100 slower than the linear evaluators described previously. We have therefore investigated an alternative Monte-Carlo algorithm, using so-called "truncated rollouts." In this technique trials are not played out to completion, but instead only a few steps in the simulation are taken, and the neural net's equity estimate of the final position reached is used instead of the actual outcome. The truncated rollout algorithm requires much less CPU time, due to two factors: First, there are potentially many

Policy iteration, which will be viewed here as the repeated use of rollout, is more ambitious and challenging than rollout. It requires off-line training, possibly in conjunction with the use of neural networks. Together with its neural network and distributed implementations, it will be discussed in Chapters 4 and 5. Unconventional forms of PI, such as multistep, simplified, and multiagent, their approximate and distributed versions, and their applications in MPC and discrete optimization are among the principal research contributions of this book. References for rollout, PI, and their variants will be given at the end of Chapters 2-5.

There is a vast literature on linear quadratic problems. The connection of PI with Newton's method within this context and its quadratic convergence rate was first derived by Kleinman [Kle68] for continuous-time problems (the corresponding discrete-time result was given by Hewer [Hew71]). For followup work, which relates to PI with approximations, see Feitzinger, Hylla, and Sachs [FHS09], and Hylla [Hyl11]. Simulation-based PI was applied to adaptive control in the context of linear quadratic problems by Vrabie, Pastravanu, Abu-Khalaf, and Lewis [VPA09], and has been discussed by several other authors. It can be used in conjunction with simulation-based model-free implementations of PI for linear quadratic problems introduced by Bradtke, Ydstie, and Barto [BYB94].

Section 1.4.4: The theory of POMDP has a long history, which dates to the 1960s. A key fact is that a POMDP can be transformed into an equivalent perfect state information stochastic problem, by using the belief state or other sufficient statistics (see, e.g., [Ber17a], Ch. 4). This brings to bear perfect state information DP, as well as state estimation methodologies, such as Kalman filtering, which is a broad subject with many applications.

Another interesting fact is that a perfect state information stochastic problem (and hence also a POMDP) can be transformed into an equivalent deterministic problem defined over a space of probability distributions. This is a conceptually useful idea. In fact it is the foundation for the mathematical development of the book by Bertsekas and Shreve [BeS78], which deals with measurability issues in stochastic DP. Within the RL methodology, the POMDP to deterministic problem transformation has been exploited by Ng and Jordan [NgJ13].

Section 1.4.5: Multiagent problems have a long history (Marschak [Mar55], Radner [Rad62], Witsenhausen [Wit68], [Wit71a], [Wit71b]), and were re-

---

fewer steps per trial. Second, there is much less variance per trial, since only a few random steps are taken and a real-valued estimate is recorded, rather than many random steps and an integer final outcome. These two factors combine to give at least an order of magnitude speed-up compared to full rollouts, while still giving a large error reduction relative to the base player." Analysis and computational experience with truncated rollout since 1996 are consistent with the preceding assessment.

searched extensively in the 70s; see the review paper by Ho [Ho80] and the references cited there. The names used for the field at that time were *team theory* and *decentralized control*. For a sampling of subsequent works in team theory and multiagent optimization, we refer to the papers by Krainak, Speyer, and Marcus [KLM82a], [KLM82b], and de Waal and van Schuppen [WaS00]. For more recent works, see Nayyar, Mahajan, and Teneketzis [NMT13], Nayyar and Teneketzis [NaT19], Li et al. [LTZ19], Qu and Li [QuL19], Gupta [Gup20], the book by Zoppoli, Sanguineti, Gnecco, and Parisini [ZSG20], and the references quoted there. Surveys of multiagent sequential decision making from an RL perspective were given by Busoniu, Babuska, and De Schutter [BBD08], [BBD10b].

We note that the term "multiagent" has been used with several different meanings in the literature. For example some authors place emphasis on the case where the agents do not have common information when selecting their decisions. This gives rise to sequential decision problems with "nonclassical information patterns," which can be very complex, partly because they cannot be addressed by exact DP. Other authors adopt as their starting point a problem where the agents are "weakly" coupled through the system equation, the cost function, or the constraints, and consider methods that exploit the weak coupling to address the problem through (suboptimal) decoupled computations.

Agent-by-agent minimization in multiagent approximation in value space and rollout was first proposed in the author's paper [Ber19c], which also discusses extensions to infinite horizon PI algorithms, and explores connections with the concept of person-by-person optimality from team theory; see also the survey [Ber21a], and the papers [Ber19d], [Ber20]. A computational study where several of the multiagent algorithmic ideas were tested and validated is the paper by Bhattacharya et al. [BKB20]. This paper considers a large-scale multi-robot routing and repair problem, involving partial state information, and explores some of the attendant implementation issues, including autonomous multiagent rollout, through the use of policy neural networks and other precomputed signaling policies.

Applications of rollout in multiagent problems and MPC will be discussed in Chapter 3. Rollout will also be applied to deterministic discrete spaces problems, including a vast array of combinatorial optimization problems in Chapter 3. Some of these applications involve multiagent rollout.

Sections 1.4.6-1.4.7: Adaptive control is a classical subject with an extensive literature, including quite a few books, which we list alphabetically: Astolfi, Karagiannis, and Ortega [AKO07], Aström and Hagglund [AsH95], [AsH06], Aström and Murray [AsM10], Aström and Wittenmark [AsW08], Bodson [Bod20], Goodwin and Sin [GoS84], Ioannou and Sun [IoS96], Jiang and Jiang [JiJ17], Krstic, Kanellakopoulos, and Kokotovic [KKK95], Kokotovic [Kok91], Kumar and Varaiya [KuV86], Liu, et al. [LWW17], Lavretsky and Wise [LaW13], Narendra and Annaswamy [NaA12], Sastry and Bod-

son [SaB11], Slotine and Li [SlL91], and Vrabie, Vamvoudakis, and Lewis [VVL13].

These books describe a vast array of methods spanning 60 years, and ranging from adaptive and PID model-free approaches, to simultaneous or sequential control and identification, to ARMAX/time series models, to extremum-seeking methods, to simulation-based RL techniques, etc.†

The research on problems involving unknown models and using data for model identification prior to or simultaneously with control was rekindled with the advent of the artificial intelligence side of RL and its focus on the active exploration of the environment. Here there is emphasis in "learning from interaction with the environment" [SuB18] through the use of (possibly hidden) Markov decision models, machine learning, and neural networks, in a wide array of methods that are under active development at present. In this book we will not deal with unknown models and the attendant system identification issues, except tangentially, and in the context of on-line replanning for problems with changing model parameters.

The idea of using simulation-based rollout for on-line indirect adaptive control (cf. Section 1.4.7), as a more economical substitute for reoptimization, seems to be new and applies to a broader class of problems than the ones addressed with traditional adaptive control methods. In particular, our approach applies under no restrictions on the state and control spaces, which can be discrete or continuous or a mixture thereof.

The literature on MPC is voluminous. Some early widely cited papers are Clarke, Mohtadi, and Tuffs [CMT87a], [CMT87b], and Keerthi and Gilbert [KeG88]. For early surveys, see Morari and Lee [MoL99], Mayne et al. [MRR00], and Findeisen et al. [FIA03], and for a more recent review, see Mayne [May14]. The connections between MPC and rollout were discussed in the author's survey [Ber05a]. Textbooks on MPC include Maciejowski [Mac02], Goodwin, Seron, and De Dona [GSD06], Camacho and Bordons [CaB07], Kouvaritakis and Cannon [KoC16], Borrelli, Bemporad, and Morari [BBM17], and Rawlings, Mayne, and Diehl [RMD17].

The extensive experience with MPC schemes has confirmed their beneficial robustness properties in the face of changing system parameters. In view of the connection between MPC and rollout, this supports the idea of adaptive control by rollout and on-line replanning.

From AlphaZero and TD-Gammon to Optimal, Adaptive, and Model Predictive Control

While the ideas of rollout and PI go back many years, their significance has been highlighted by the success of AlphaZero and the earlier, but just

† The ideas of PID control originated even earlier, nearly 100 years ago. According to Wikipedia, "a formal control law for what we now call PID or three-term control was first developed using theoretical analysis, by Russian American engineer Nicolas Minorsky" [Min22].

as impressive, TD-Gammon program (more accurately the 1996 rollout version of the program [BeG96], which uses short lookahead minimization, long truncated rollout, and terminal cost approximation).

Both programs were trained off-line extensively using sophisticated approximate PI algorithms and neural networks. Yet the players obtained off-line were greatly improved by on-line play. In particular:

(a) The on-line player of AlphaZero plays much better than its extensively trained off-line player. This is due to the beneficial effect of approximation in value space with long lookahead minimization, which corrects for the inevitable imperfections of the off-line player and its value network.

(b) The TD-Gammon player that uses long truncated rollout between short lookahead minimization and terminal cost approximation plays much better than TD-Gammon without rollout. This is due to the beneficial effect of long rollout, which serves as a substitute for long lookahead minimization.

In practice, whether in the context of games or decision and control, a lot of analysis and/or off-line computation is often directed towards obtaining a policy, which is inevitably suboptimal, because of policy representation errors, model imperfections, changing problem parameters, and overwhelming computational bottlenecks. An important lesson from AlphaZero and TD-Gammon is that performance may be greatly improved by on-line approximation in value space, with long lookahead (whether involving minimization or rollout with an off-line obtained policy), and possibly terminal cost approximation that may be obtained off-line.

This performance enhancement by on-line play goes well beyond the conventional control wisdom that "feedback corrects for noise, uncertainty, and modeling errors." It defines a new paradigm, whose implications have yet to be fully appreciated within the decision and control community.

There is an additional benefit of policy improvement by approximation in value space, not observed in the context of games (which have stable rules and environment). It is well-suited for on-line replanning and changing problem parameters, as in the context of indirect adaptive control.

In this book we aim to provide the mathematical framework, analysis, and insights (often based on visualization), which facilitate the use of on-line decision making on top of off-line training. In particular, through a unified analysis, we show that the principal ideas of approximation in value space and rollout apply very broadly to deterministic, stochastic, and combinatorial/integer problems, involving both discrete and continuous search spaces. Moreover, these ideas can be effectively integrated with other important methodologies such as MPC, adaptive control, decentralized control, discrete and Bayesian optimization, neural network-based value and policy approximations, and heuristic algorithms for discrete optimization.

Research Content of this Book

This book has a research focus and a point of view. It provides a synthesis of most of the research of the author and his collaborators on rollout, PI algorithms, and related RL methods, starting with the rollout papers [BTW97] and [BeC98]. The original sources, the nature of the contributions, and the relation to preexisting works are recounted in the end-of-chapter references. Recent work of the author, published in the last couple of years, has focused primarily on multiagent and distributed computation issues [Ber19c], [Ber19d], [Ber20], [Ber21a], [BBW20], and [BKB20], as well as on-line PI [Ber21b]. It is discussed at length in this book. Several other research ideas, which were developed as the book was being written, are sprinkled throughout the text and have been noted at the appropriate points.

Beyond the details and analysis of individual algorithms, the book aims to convey the author's view that the rollout approach, coupled with on-line approximation in value space and multistep lookahead, is theoretically solid, thanks to its cost improvement property, and its relation to PI and Newton's method. Most importantly, based on extensive computational experience, rollout and its variations seem to be by far the most reliable and easiest to implement RL algorithmic approach.

E X E R C I S E S

1.1 (Computational Exercise - Multi-Vehicle Routing)

In the routing problem of Example 1.2.3 view the two vehicles as separate agents.

(a) Apply the multiagent rollout algorithm of Section 1.4.5, starting from the pair location (1,2). Compare the multiagent rollout performance with the performance of the ordinary rollout algorithm, where both vehicles move at once, and with the base heuristic that moves each vehicle one step along the shortest path to the nearest pending task.

(b) Repeat for the case where, in addition to vehicles 1 and 2 that start at nodes 1 and 2, respectively, there is a third vehicle that starts at node 10, and there is a third task to be performed at node 8.

1.2 (Computational Exercise - Spiders and Flies)

Consider the spiders and flies problem of Example 1.2.3 with two differences: the five flies are stationary (rather than moving randomly), and there are only two spiders that start at the fourth square from the right at the top row of the grid of Fig. 1.4.7. The base heuristic is to move each spider one square towards its nearest fly, with distance measured by the Manhattan metric, and with preference

given to a horizontal direction over a vertical direction in case of a tie. Apply the multiagent rollout algorithm of Section 1.4.5, and compare its performance with the one of the ordinary rollout algorithm, and with the one of the base heuristic.

1.3 (Computational Exercise - Linear Quadratic Problem)

In a more realistic version of the cruise control system of Example 1.3.1, the system has the form $x_{k+1} = ax_k + bu_k + w_k$, where the coefficient a satisfies $0 < a \leq 1$, and the disturbance w_k has zero mean and variance σ^2. The cost function has the form

$$(x_N - \bar{x}_N)^2 + \sum_{k=0}^{N-1} \big((x_k - \bar{x}_k)^2 + ru_k^2\big),$$

where $\bar{x}_0, \ldots, \bar{x}_N$ are given nonpositive target values (a velocity profile) that serve to adjust the vehicle's velocity, in order to maintain a safe distance from the vehicle ahead, etc. In a practical setting, the velocity profile is recalculated by using on-line radar measurements.

(a) Use exact DP to derive the optimal policy for given values of a and b, and a given velocity profile. *Hint*: Hypothesize that $J_k^*(x_k)$, the optimal cost-to-go starting from state x_k at time k, has the form

$$J_k^*(x_k) = P_k x_k^2 - 2p_k x_k + \text{constant},$$

and assuming your hypothesis is correct, verify that the optimal policy is linear in the state plus a constant, and has the form $\mu_k^*(x_k) = L_k x_k + \ell_k$, where

$$L_k = -\frac{abP_{k+1}}{r + b^2 P_{k+1}}, \qquad \ell_k = \frac{bp_{k+1}}{r + b^2 P_{k+1}}.$$

Use the DP algorithm to verify that your hypothesis is correct, and to derive recursive formulas that express the scalar coefficients P_k and p_k in terms of P_{k+1}, p_{k+1}, L_k, and ℓ_k, starting with $P_N = 1$ and $p_N = \bar{x}_N$.

(b) Design an experiment to compare the performance of a fixed linear policy π, derived for a fixed nominal velocity profile as in part (a), and the performance of the algorithm that uses on-line replanning, whereby the optimal policy π^* is recalculated each time the velocity profile changes. Compare with the performance of the rollout policy $\tilde{\pi}$ that uses π as the base policy and on-line replanning.

1.4 (Computational Exercise - Parking Problem)

In reference to Example 1.4.3, a driver aims to park at an inexpensive space on the way to his destination. There are L parking spaces available and a garage at the end. The driver can move in either direction. For example if he is in space i he can either move to $i-1$ with a cost t_i^- , or to $i+1$ with a cost t_i^+, or he can park at a cost $c(i)$ (if the parking space i is free). The only exception is when he

arrives at the garage (indicated by index N) and he has to park there at a cost C. Moreover, after the driver visits a parking space he remembers its free/taken status and has an option to return to any parking space he has already visited. However, the driver must park within a given number of stages N, so that the problem has a finite horizon. The initial probability of each space being free is given, and the driver can only observe the free/taken status of a parking space only after he/she visits the space. Moreover, the free/taken status of a parking space that has been visited so far does not change over time.

Write a program to calculate the optimal solution using exact dynamic programming over a state space that is as small as possible. Try to experiment with different problem data, and try to visualize the optimal cost/policy with suitable graphical plots. Comment on the run-time as you increase the number of parking spots L.

1.5 (PI and Newton's Method for Linear Quadratic Problems)

The purpose of this exercise is to demonstrate the fast convergence of the PI algorithm for the case of the one-dimensional linear quadratic problem without discounting ($\alpha = 1$); cf. Examples 1.3.5 and 1.4.6.

(a) Verify that the Bellman equation,

$$Kx^2 = \min_u \left[x^2 + ru^2 + K(x + bu)^2\right],$$

can be written as the equation $H(K) = 0$, where $H(K) = K - \frac{rK}{r+b^2K} - 1$.

(b) Consider the PI algorithm

$$K_k = \frac{1 + rL_k^2}{1 - (1 + bL_k)^2},$$

where

$$L_{k+1} = -\frac{bK_k}{r + b^2K_k},$$

and $\mu(x) = L_0x$ is the starting policy. Verify that this iteration is equivalent to Newton's method

$$K_{k+1} = K_k - \left(\frac{\partial H(K_k)}{\partial K}\right)^{-1} H(K_k),$$

for solving the Bellman equation $H(K) = 0$. This relation can be shown in greater generality, for multidimensional linear quadratic problems.

(c) Verify computationally that $\lim_{J \to J^*} \frac{\tilde{J} - J^*}{J - J^*} = 0$, for the two cases where $r = 0.5$ and b varies, and $b = 2$ and r varies. Here for a given initial state, $\tilde{J}$ is the rollout performance, J^* is the optimal performance, and J is the performance of the base policy $\mu_L(x) = Lx$, where L is given by Eq. (1.64).

(d) *Rollout cost improvement over base policy in adaptive control*: Consider a range of values of b, r, and L_0, and study computationally the effect of the

second derivative of H on the ratio K_1/K_0 of rollout to base policy costs, and on the ratio K_1/K^* of rollout to optimal policy costs.

Solution of part (b): We have $L = -\frac{bK}{r+b^2K}$ if and only if $K = -\frac{rL}{b(1+bL)}$, so

$$H(K) = K - \frac{rK}{r+b^2K} - 1 = \frac{b^2K^2}{r+b^2K} - 1 = -bLK - 1 = \frac{rL^2 - (bL+1)}{bL+1}.$$

We next calculate the inverse partial derivative $\frac{\partial H}{\partial K}$ as

$$\begin{aligned}\Big(\frac{\partial H}{\partial K}\Big)^{-1} &= \frac{(r+b^2K)^2}{b^2K(2r+b^2K)} = \frac{1}{L^2}\cdot\frac{K}{2r+b^2K} = \frac{1}{L^2}\cdot\frac{1+bL}{2+bL}\cdot\frac{K}{r}\\ &= -\frac{1}{L^2}\cdot\frac{1+bL}{2+bL}\cdot\frac{L}{b(1+bL)} = -\frac{1}{bL(2+bL)}.\end{aligned}$$

Therefore, the iterate of Newton's method is given by

$$\begin{aligned}K - \Big(\frac{\partial H}{\partial K}\Big)^{-1} H(K) &= -\frac{rL}{b(1+bL)} + \frac{1}{bL(2+bL)}\cdot\frac{rL^2-(bL+1)}{1+bL}\\ &= \frac{-rL^2(2+bL)+rL^2-(1+bL)}{bL(2+bL)(1+bL)}\\ &= \frac{-rL^2(1+bL)-(1+bL)}{bL(2+bL)(1+bL)} = \frac{-rL^2-1}{bL(2+bL)} = \frac{1+rL^2}{1-(1+bL)^2},\end{aligned}$$

and is identical to the PI iterate.

1.6 (Linear Quadratic Problems: Stability in Approximation in Value Space and Adaptive Control)

This exercise focuses on the linear quadratic problem of Examples 1.3.3 and 1.4.6, and aims to address issues of stability.

(a) Consider approximation in value space with $\tilde{J}(x) = \beta x^2$. Show that for $\beta = 0$, the closed-loop system obtained from the one-step lookahead minimization is $x_{k+1} = x_k$, so it is unstable.

(b) Consider the issue of stability of the closed-loop system under the base and the rollout policies, as the problem parameters b and r change. Let the base policy be $\mu(x) = Lx$, where L is given by Eq. (1.64). This base policy is stable when $|1+bL| < 1$, or equivalently if $0 < b < \frac{2(5+2\sqrt{6})}{2+\sqrt{6}}$. The rollout policy is stable when $|1+b\tilde{L}| < 1$, where $\tilde{L}$ is given by Eq. (1.66). Verify that this is equivalent to $0 < 2r + b^2K_L$, where K_L is given by Eq. (1.65), and conclude that contrary to the base policy, the rollout policy is stable for all values of b and values of $r > 0$.

(c) Suppose that the system is $x_{k+1} = ax_k + bu_k$ instead of $x_{k+1} = x_k + bu_k$. Derive the range of values of a for which the conclusion of part (b) holds. Explain your results with the aid of Figs. 1.3.10 and 1.3.12.

Note: The issue of stability of policies obtained by approximation in value space (including MPC) is important, and is intimately connected with the notion of *sequential improvement* conditions (also called Lyapunov conditions); see Sections 2.3 and 3.1.2, and Exercises 5.1-5.3 in Chapter 5, for further discussion.

References

[ABB19] Agrawal, A., Barratt, S., Boyd, S., and Stellato, B., 2019. "Learning Convex Optimization Control Policies," arXiv preprint arXiv:1912.09529.

[ACF02] Auer, P., Cesa-Bianchi, N., and Fischer, P., 2002. "Finite Time Analysis of the Multiarmed Bandit Problem," Machine Learning, Vol. 47, pp. 235-256.

[ADH19] Arora, S., Du, S. S., Hu, W., Li, Z., and Wang, R., 2019. "Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks," arXiv preprint arXiv:1901.08584.

[AHZ19] Arcari, E., Hewing, L., and Zeilinger, M. N., 2019. "An Approximate Dynamic Programming Approach for Dual Stochastic Model Predictive Control," arXiv preprint arXiv:1911.03728.

[AKO07] Astolfi, A., Karagiannis, D., and Ortega, R., 2007. Nonlinear and Adaptive Control with Applications, Springer, N. Y.

[ALZ08] Asmuth, J., Littman, M. L., and Zinkov, R., 2008. "Potential-Based Shaping in Model-Based Reinforcement Learning," Proc. of 23rd AAAI Conference, pp. 604-609.

[AMS09] Audibert, J.Y., Munos, R., and Szepesvari, C., 2009. "Exploration-Exploitation Tradeoff Using Variance Estimates in Multi-Armed Bandits," Theoretical Computer Science, Vol. 410, pp. 1876-1902.

[ASR20] Andersen, A. R., Stidsen, T. J. R., and Reinhardt, L. B., 2020. "Simulation-Based Rolling Horizon Scheduling for Operating Theatres," in SN Operations Research Forum, Vol. 1, pp. 1-26.

[AXG16] Ames, A. D., Xu, X., Grizzle, J. W., and Tabuada, P., 2016. "Control Barrier Function Based Quadratic Programs for Safety Critical Systems," IEEE Trans. on Automatic Control, Vol. 62, pp. 3861-3876.

[Abr90] Abramson, B., 1990. "Expected-Outcome: A General Model of Static Evaluation," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 12, pp. 182-193.

[Agr95] Agrawal, R., 1995. "Sample Mean Based Index Policies with $O(\log n)$ Regret for the Multiarmed Bandit Problem," Advances in Applied Probability, Vol. 27, pp. 1054-1078.

[AnH14] Antunes, D., and Heemels, W. P. M. H., 2014. "Rollout Event-Triggered Control: Beyond Periodic Control Performance," IEEE Trans. on Automatic Control, Vol. 59, pp. 3296-3311.

[AnM79] Anderson, B. D. O., and Moore, J. B., 1979. Optimal Filtering, Prentice-Hall, Englewood Cliffs, N. J.

[AsH95] Aström, K. J., and Hagglund, T., 1995. PID Controllers: Theory, Design, and Tuning, Instrument Society of America, Research Triangle Park, NC.

[AsH06] Aström, K. J., and Hagglund, T., 2006. Advanced PID Control, Instrument Society of America, Research Triangle Park, N. C.

[AsM10] Aström, K. J., and Murray, R. M., 2010. Feedback Systems, Princeton Univ. Press.

[AsW08] Aström, K. J., and Wittenmark, B., 2008. Adaptive Control, Dover Books; also Prentice-Hall, Englewood Cliffs, N. J, 1994.

[Ast83] Aström, K. J., 1983. "Theory and Applications of Adaptive Control - A Survey," Automatica, Vol. 19, pp. 471-486.

[AtF66] Athans, M., and Falb, P., 1966. Optimal Control, McGraw-Hill, N. Y.

[AvB20] Avrachenkov, K., and Borkar, V. S., 2020. "Whittle Index Based Q-Learning for Restless Bandits with Average Reward," arXiv preprint arXiv:2004.14427.

[BBD08] Busoniu, L., Babuska, R., and De Schutter, B., 2008. "A Comprehensive Survey of Multiagent Reinforcement Learning," IEEE Trans. on Systems, Man, and Cybernetics, Part C, Vol. 38, pp. 156-172.

[BBD10a] Busoniu, L., Babuska, R., De Schutter, B., and Ernst, D., 2010. Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press, N. Y.

[BBD10b] Busoniu, L., Babuska, R., and De Schutter, B., 2010. "Multi-Agent Reinforcement Learning: An Overview," in Innovations in Multi-Agent Systems and Applications, Springer, pp. 183-221.

[BBG13] Bertazzi, L., Bosco, A., Guerriero, F., and Lagana, D., 2013. "A Stochastic Inventory Routing Problem with Stock-Out," Transportation Research, Part C, Vol. 27, pp. 89-107.

[BBK20] Balandin, D. V., Biryukov, R. S., and Kogan, M. M., 2020. "Ellipsoidal Reachable Sets of Linear Time-Varying Continuous and Discrete Systems in Control and Estimation Problems,"Automatica, Vol. 116, p. 108926.

[BBM17] Borrelli, F., Bemporad, A., and Morari, M., 2017. Predictive Control for Linear and Hybrid Systems, Cambridge Univ. Press, Cambridge, UK.

[BBP13] Bhatnagar, S., Borkar, V. S., and Prashanth, L. A., 2013. "Adaptive Feature Pursuit: Online Adaptation of Features in Reinforcement Learning," in *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, by F. Lewis and D. Liu (eds.), IEEE Press, Piscataway, N. J., pp. 517-534.

[BBW20] Bhattacharya, S., Badyal, S., Wheeler, T., Gil, S., and Bertsekas, D. P., 2020. "Reinforcement Learning for POMDP: Partitioned Rollout and Policy Iteration with Application to Autonomous Sequential Repair Problems," IEEE Robotics and Automation Letters, Vol. 5, pp. 3967-3974.

[BCD10] Brochu, E., Cora, V. M., and De Freitas, N., 2010. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," arXiv preprint arXiv:1012.2599.

[BCN18] Bottou, L., Curtis, F. E., and Nocedal, J., 2018. "Optimization Methods for Large-Scale Machine Learning," SIAM Review, Vol. 60, pp. 223-311.

[BKB20] Bhattacharya, S., Kailas, S., Badyal, S., Gil, S., and Bertsekas, D. P., 2020. "Multiagent Rollout and Policy Iteration for POMDP with Application to Multi-Robot Repair Problems," in Proc. of Conference on Robot Learning (CoRL); also arXiv preprint,

arXiv:2011.04222.

[BLL19] Bartlett, P. L., Long, P. M., Lugosi, G., and Tsigler, A., 2019. "Benign Overfitting in Linear Regression," arXiv preprint arXiv:1906.11300.

[BMM18] Belkin, M., Ma, S., and Mandal, S., 2018. "To Understand Deep Learning we Need to Understand Kernel Learning," arXiv preprint arXiv:1802.01396.

[BMZ09] Bokanowski, O., Maroso, S., and Zidani, H., 2009. "Some Convergence Results for Howard's Algorithm," SIAM J. on Numerical Analysis, Vol. 47, pp. 3001-3026.

[BPW12] Browne, C., Powley, E., Whitehouse, D., Lucas, L., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S., 2012. "A Survey of Monte Carlo Tree Search Methods," IEEE Trans. on Computational Intelligence and AI in Games, Vol. 4, pp. 1-43.

[BRT18] Belkin, M., Rakhlin, A., and Tsybakov, A. B., 2018. "Does Data Interpolation Contradict Statistical Optimality?" arXiv preprint arXiv:1806.09471.

[BSW99] Beard, R. W., Saridis, G. N., and Wen, J. T., 1998. "Approximate Solutions to the Time-Invariant Hamilton-Jacobi-Bellman Equation," J. of Optimization Theory and Applications, Vol. 96, pp. 589-626.

[BTW97] Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C., 1997. "Rollout Algorithms for Combinatorial Optimization," Heuristics, Vol. 3, pp. 245-262.

[BWL19] Beuchat, P. N., Warrington, J., and Lygeros, J., 2019. "Accelerated Point-Wise Maximum Approach to Approximate Dynamic Programming," arXiv preprint arXiv:1901.03619.

[BYB94] Bradtke, S. J., Ydstie, B. E., and Barto, A. G., 1994. "Adaptive Linear Quadratic Control Using Policy Iteration," Proc. IEEE American Control Conference, Vol. 3, pp. 3475-3479.

[BaF88] Bar-Shalom, Y., and Fortman, T. E., 1988. Tracking and Data Association, Academic Press, N. Y.

[BaL19] Banjac, G., and Lygeros, J., 2019. "A Data-Driven Policy Iteration Scheme Based on Linear Programming," Proc. 2019 IEEE CDC, pp. 816-821.

[BaM20] Barzgaran, B., and Mesbahi, M., 2020. "Robust Analysis and Sensitivity Design of Model Predictive Control,"IFAC-PapersOnLine,Vol. 53, pp. 7111-7116.

[BaP12] Bauso, D., and Pesenti, R., 2012. "Team Theory and Person-by-Person Optimization with Binary Decisions," SIAM J. on Control and Optimization, Vol. 50, pp. 3011-3028.

[Bad21] Badyal, S., 2021. A Comparative Study of Multi-Agent Reinforcement Learning on Real World Problems, MS Thesis, Arizona State University.

[Bai93] Baird, L. C., 1993. "Advantage Updating," Report WL-TR-93-1146, Wright Patterson AFB, OH.

[Bai94] Baird, L. C., 1994. "Reinforcement Learning in Continuous Time: Advantage Updating," International Conf. on Neural Networks, Orlando, Fla.

[Bar90] Bar-Shalom, Y., 1990. Multitarget-Multisensor Tracking: Advanced Applications, Artech House, Norwood, MA.

[BeC89] Bertsekas, D. P., and Castañon, D. A., 1989. "The Auction Algorithm for Transportation Problems," Annals of Operations Research, Vol. 20, pp. 67-96.

[BeC99] Bertsekas, D. P., and Castañon, D. A., 1999. "Rollout Algorithms for Stochastic Scheduling Problems," Heuristics, Vol. 5, pp. 89-108.

[BeC02] Ben-Gal, I., and Caramanis, M., 2002. "Sequential DOE via Dynamic Programming," IIE Transactions, Vol. 34, pp. 1087-1100.

[BeC08] Besse, C., and Chaib-draa, B., 2008. "Parallel Rollout for Online Solution of DEC-POMDPs," Proc. of 21st International FLAIRS Conference, pp. 619-624.

[BeI96] Bertsekas, D. P., and Ioffe, S., 1996. "Temporal Differences-Based Policy Iteration and Applications in Neuro-Dynamic Programming," Lab. for Info. and Decision Systems Report LIDS-P-2349, Massachusetts Institute of Technology.

[BeL14] Beyme, S., and Leung, C., 2014. "Rollout Algorithm for Target Search in a Wireless Sensor Network," 80th Vehicular Technology Conference (VTC2014), IEEE, pp. 1-5.

[BeP03] Bertsimas, D., and Popescu, I., 2003. "Revenue Management in a Dynamic Network Environment," Transportation Science, Vol. 37, pp. 257-277.

[BeR71a] Bertsekas, D. P., and Rhodes, I. B., 1971. "On the Minimax Reachability of Target Sets and Target Tubes," Automatica, Vol. 7, pp. 233-247.

[BeR71b] Bertsekas, D. P., and Rhodes, I. B., 1971. "Recursive State Estimation for a Set-Membership Description of the Uncertainty," IEEE Trans. Automatic Control, Vol. AC-16, pp. 117-128.

[BeR73] Bertsekas, D. P., and Rhodes, I. B., 1973. "Sufficiently Informative Functions and the Minimax Feedback Control of Uncertain Dynamic Systems," IEEE Trans. Automatic Control, Vol. AC-18, pp. 117-124.

[BeS78] Bertsekas, D. P., and Shreve, S. E., 1978. Stochastic Optimal Control: The Discrete Time Case, Academic Press, N. Y.; republished by Athena Scientific, Belmont, MA, 1996 (can be downloaded from the author's website).

[BeS18] Bertazzi, L., and Secomandi, N., 2018. "Faster Rollout Search for the Vehicle Routing Problem with Stochastic Demands and Restocking," European J. of Operational Research, Vol. 270, pp.487-497.

[BeT89] Bertsekas, D. P., and Tsitsiklis, J. N., 1989. Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Englewood Cliffs, N. J.; republished by Athena Scientific, Belmont, MA, 1997 (can be downloaded from the author's website).

[BeT91] Bertsekas, D. P., and Tsitsiklis, J. N., 1991. "An Analysis of Stochastic Shortest Path Problems," Math. Operations Res., Vol. 16, pp. 580-595.

[BeT96] Bertsekas, D. P., and Tsitsiklis, J. N., 1996. Neuro-Dynamic Programming, Athena Scientific, Belmont, MA.

[BeT97] Bertsimas, D., and Tsitsiklis, J. N., 1997. Introduction to Linear Optimization, Athena Scientific, Belmont, MA.

[BeT00] Bertsekas, D. P., and Tsitsiklis, J. N., 2000. "Gradient Convergence of Gradient Methods with Errors," SIAM J. on Optimization, Vol. 36, pp. 627-642.

[BeT08] Bertsekas, D. P., and Tsitsiklis, J. N., 2008. Introduction to Probability, 2nd Ed., Athena Scientific, Belmont, MA.

[BeY09] Bertsekas, D. P., and Yu, H., 2009. "Projected Equation Methods for Approximate Solution of Large Linear Systems," J. of Computational and Applied Math., Vol. 227, pp. 27-50.

[BeY10] Bertsekas, D. P., and Yu, H., 2010. "Asynchronous Distributed Policy Iteration

in Dynamic Programming," Proc. of Allerton Conf. on Communication, Control and Computing, Allerton Park, Ill, pp. 1368-1374.

[BeY12] Bertsekas, D. P., and Yu, H., 2012. "Q-Learning and Enhanced Policy Iteration in Discounted Dynamic Programming," Math. of Operations Research, Vol. 37, pp. 66-94.

[BeY16] Bertsekas, D. P., and Yu, H., 2016. "Stochastic Shortest Path Problems Under Weak Conditions," Lab. for Information and Decision Systems Report LIDS-2909, Massachusetts Institute of Technology.

[Bea95] Beard, R. W., 1995. Improving the Closed-Loop Performance of Nonlinear Systems, Ph.D. Thesis, Rensselaer Polytechnic Institute.

[Bel56] Bellman, R., 1956. "A Problem in the Sequential Design of Experiments," Sankhya: The Indian J. of Statistics, Vol. 16, pp. 221-229.

[Bel57] Bellman, R., 1957. Dynamic Programming, Princeton University Press, Princeton, N. J.

[Bel67] Bellman, R., 1967. Introduction to the Mathematical Theory of Control Processes, Academic Press, Vols. I and II, New York, N. Y.

[Bel84] Bellman, R., 1984. Eye of the Hurricane, World Scientific Publishing, Singapore.

[Ben09] Bengio, Y., 2009. "Learning Deep Architectures for AI," Foundations and Trends in Machine Learning, Vol. 2, pp. 1-127.

[Ber71] Bertsekas, D. P., 1971. "Control of Uncertain Systems With a Set-Membership Description of the Uncertainty," Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA (can be downloaded from the author's website).

[Ber72] Bertsekas, D. P., 1972. "Infinite Time Reachability of State Space Regions by Using Feedback Control," IEEE Trans. Automatic Control, Vol. AC-17, pp. 604-613.

[Ber73] Bertsekas, D. P., 1973. "Linear Convex Stochastic Control Problems over an Infinite Horizon," IEEE Trans. Automatic Control, Vol. AC-18, pp. 314-315.

[Ber77] Bertsekas, D. P., 1977. "Monotone Mappings with Application in Dynamic Programming," SIAM J. on Control and Opt., Vol. 15, pp. 438-464.

[Ber79] Bertsekas, D. P., 1979. "A Distributed Algorithm for the Assignment Problem," Lab. for Information and Decision Systems Report, Massachusetts Institute of Technology, May 1979.

[Ber82] Bertsekas, D. P., 1982. "Distributed Dynamic Programming," IEEE Trans. Automatic Control, Vol. AC-27, pp. 610-616.

[Ber83] Bertsekas, D. P., 1983. "Asynchronous Distributed Computation of Fixed Points," Math. Programming, Vol. 27, pp. 107-120.

[Ber91] Bertsekas, D. P., 1991. Linear Network Optimization: Algorithms and Codes, MIT Press, Cambridge, MA (can be downloaded from the author's website).

[Ber96] Bertsekas, D. P., 1996. "Incremental Least Squares Methods and the Extended Kalman Filter," SIAM J. on Optimization, Vol. 6, pp. 807-822.

[Ber97a] Bertsekas, D. P., 1997. "A New Class of Incremental Gradient Methods for Least Squares Problems," SIAM J. on Optimization, Vol. 7, pp. 913-926.

[Ber97b] Bertsekas, D. P., 1997. "Differential Training of Rollout Policies," Proc. of the 35th Allerton Conference on Communication, Control, and Computing, Allerton Park, Ill.

[Ber98] Bertsekas, D. P., 1998. Network Optimization: Continuous and Discrete Models, Athena Scientific, Belmont, MA (can be downloaded from the author's website).

[Ber05a] Bertsekas, D. P., 2005. "Dynamic Programming and Suboptimal Control: A Survey from ADP to MPC," European J. of Control, Vol. 11, pp. 310-334.

[Ber05b] Bertsekas, D. P., 2005. "Rollout Algorithms for Constrained Dynamic Programming," Lab. for Information and Decision Systems Report LIDS-P-2646, Massachusetts Institute of Technology.

[Ber07] Bertsekas, D. P., 2007. "Separable Dynamic Programming and Approximate Decomposition Methods," IEEE Trans. on Aut. Control, Vol. 52, pp. 911-916.

[Ber10a] Bertsekas, D. P., 2010. "Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey," Lab. for Information and Decision Systems Report LIDS-P-2848, Massachusetts Institute of Technology; a condensed version with the same title appears in Optimization for Machine Learning, by S. Sra, S. Nowozin, and S. J. Wright, (eds.), MIT Press, Cambridge, MA, 2012, pp. 85-119.

[Ber10b] Bertsekas, D. P., 2010. "Williams-Baird Counterexample for Q-Factor Asynchronous Policy Iteration,"
http://web.mit.edu/dimitrib/www/Williams-Baird Counterexample.pdf.

[Ber11a] Bertsekas, D. P., 2011. "Incremental Proximal Methods for Large Scale Convex Optimization," Math. Programming, Vol. 129, pp. 163-195.

[Ber11b] Bertsekas, D. P., 2011. "Approximate Policy Iteration: A Survey and Some New Methods," J. of Control Theory and Applications, Vol. 9, pp. 310-335; a somewhat expanded version appears as Lab. for Info. and Decision Systems Report LIDS-2833, Massachusetts Institute of Technology, 2011.

[Ber12] Bertsekas, D. P., 2012. Dynamic Programming and Optimal Control, Vol. II, 4th Ed., Athena Scientific, Belmont, MA.

[Ber13a] Bertsekas, D. P., 2013. "Rollout Algorithms for Discrete Optimization: A Survey," Handbook of Combinatorial Optimization, Springer.

[Ber13b] Bertsekas, D. P., 2013. "λ-Policy Iteration: A Review and a New Implementation," in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, by F. Lewis and D. Liu (eds.), IEEE Press, Piscataway, N. J., pp. 381-409.

[Ber15a] Bertsekas, D. P., 2015. Convex Optimization Algorithms, Athena Scientific, Belmont, MA.

[Ber15b] Bertsekas, D. P., 2015. "Incremental Aggregated Proximal and Augmented Lagrangian Algorithms," Lab. for Information and Decision Systems Report LIDS-P-3176, Massachusetts Institute of Technology; arXiv preprint arXiv:1507.1365936.

[Ber16] Bertsekas, D. P., 2016. Nonlinear Programming, 3rd Ed., Athena Scientific, Belmont, MA.

[Ber17b] Bertsekas, D. P., 2017. Dynamic Programming and Optimal Control, Vol. I, 4th Ed., Athena Scientific, Belmont, MA.

[Ber17b] Bertsekas, D. P., 2017. "Value and Policy Iteration in Deterministic Optimal Control and Adaptive Dynamic Programming," IEEE Trans. on Neural Networks and Learning Systems, Vol. 28, pp. 500-509.

[Ber18a] Bertsekas, D. P., 2018. Abstract Dynamic Programming, 2nd Ed., Athena Scientific, Belmont, MA (can be downloaded from the author's website).

[Ber18b] Bertsekas, D. P., 2018. "Feature-Based Aggregation and Deep Reinforcement

Learning: A Survey and Some New Implementations," Lab. for Information and Decision Systems Report, Massachusetts Institute of Technology; arXiv preprint arXiv:1804.04577; IEEE/CAA J. of Automatica Sinica, Vol. 6, 2019, pp. 1-31.

[Ber18c] Bertsekas, D. P., 2018. "Biased Aggregation, Rollout, and Enhanced Policy Improvement for Reinforcement Learning," Lab. for Information and Decision Systems Report, Massachusetts Institute of Technology; arXiv preprint arXiv:1910.02426.

[Ber18d] Bertsekas, D. P., 2018. "Proximal Algorithms and Temporal Difference Methods for Solving Fixed Point Problems," Computational Optim. Appl., Vol. 70, pp. 709-736.

[Ber19a] Bertsekas, D. P., 2019. Reinforcement Learning and Optimal Control, Athena Scientific, Belmont, MA.

[Ber19b] Bertsekas, D. P., 2019. "Robust Shortest Path Planning and Semicontractive Dynamic Programming," Naval Research Logistics, Vol. 66, pp. 15-37.

[Ber19c] Bertsekas, D. P., 2019. "Multiagent Rollout Algorithms and Reinforcement Learning," arXiv preprint arXiv:1910.00120.

[Ber19d] Bertsekas, D. P., 2019. "Constrained Multiagent Rollout and Multidimensional Assignment with the Auction Algorithm," arXiv preprint, arxiv.org/abs/2002.07407.

[Ber20] Bertsekas, D. P., 2020. "Multiagent Value Iteration Algorithms in Dynamic Programming and Reinforcement Learning," arXiv preprint, arxiv.org/abs/2005.01627; Results in Control and Optimization J., Vol. 1, 2020.

[Ber21a] Bertsekas, D. P., 2021. "Multiagent Reinforcement Learning: Rollout and Policy Iteration," IEEE/CAA J. of Automatica Sinica, Vol. 8, pp. 249-271.

[Ber21b] Bertsekas, D. P., 2021. "On-Line Policy Iteration for Infinite Horizon Dynamic Programming," arXiv preprint arXiv:2106.00746, May 2021.

[Bet10] Bethke, B. M., 2010. Kernel-Based Approximate Dynamic Programming Using Bellman Residual Elimination, Ph.D. Thesis, Massachusetts Institute of Technology.

[BiL97] Birge, J. R., and Louveaux, 1997. Introduction to Stochastic Programming, Springer, New York, N. Y.

[Bia16] Bianchi, P., 2016. "Ergodic Convergence of a Stochastic Proximal Point Algorithm," SIAM J. on Optimization, Vol. 26, pp. 2235-2260.

[Bis95] Bishop, C. M, 1995. Neural Networks for Pattern Recognition, Oxford University Press, N. Y.

[Bis06] Bishop, C. M, 2006. Pattern Recognition and Machine Learning, Springer, N. Y.

[BlG54] Blackwell, D., and Girshick, M. A., 1954. Theory of Games and Statistical Decisions, Wiley, N. Y.

[BlM08] Blanchini, F., and Miani, S., 2008. Set-Theoretic Methods in Control, Birkhauser, Boston.

[Bla86] Blackman, S. S., 1986. Multi-Target Tracking with Radar Applications, Artech House, Dehdam, MA.

[Bla99] Blanchini, F., 1999. "Set Invariance in Control – A Survey," Automatica, Vol. 35, pp. 1747-1768.

[Bod20] Bodson, M., 2020. Adaptive Estimation and Control, Independently Published.

[Bor08] Borkar, V. S., 2008. Stochastic Approximation: A Dynamical Systems Viewpoint, Cambridge Univ. Press.

[BrH75] Bryson, A., and Ho, Y. C., 1975. Applied Optimal Control: Optimization, Estimation, and Control, (revised edition), Taylor and Francis, Levittown, Penn.

[Bra21] Brandimarte, P., 2021. From Shortest Paths to Reinforcement Learning: A MATLAB-Based Tutorial on Dynamic Programming, Springer.

[BuK97] Burnetas, A. N., and Katehakis, M. N., 1997. "Optimal Adaptive Policies for Markov Decision Processes," Math. of Operations Research, Vol. 22, pp. 222-255.

[CBH09] Choi, H. L., Brunet, L., and How, J. P., 2009. "Consensus-Based Decentralized Auctions for Robust Task Allocation," IEEE Trans. on Robotics, Vol. 25, pp. 912-926.

[CHF05] Chang, H. S., Hu, J., Fu, M. C., and Marcus, S. I., 2005. "An Adaptive Sampling Algorithm for Solving Markov Decision Processes," Operations Research, Vol. 53, pp. 126-139.

[CHF13] Chang, H. S., Hu, J., Fu, M. C., and Marcus, S. I., 2013. Simulation-Based Algorithms for Markov Decision Processes, 2nd Ed., Springer, N. Y.

[CLT19] Chapman, M. P., Lacotte, J., Tamar, A., Lee, D., Smith, K. M., Cheng, V., Fisac, J. F., Jha, S., Pavone, M., and Tomlin, C. J., 2019. "A Risk-Sensitive Finite-Time Reachability Approach for Safety of Stochastic Dynamic Systems," arXiv preprint arXiv:1902.11277.

[CMT87a] Clarke, D. W., Mohtadi, C., and Tuffs, P. S., 1987. "Generalized Predictive Control - Part I. The Basic Algorithm," Automatica Vol. 23, pp. 137-148.

[CMT87b] Clarke, D. W., Mohtadi, C., and Tuffs, P. S., 1987. "Generalized Predictive Control - Part II," Automatica Vol. 23, pp. 149-160.

[CRV06] Cogill, R., Rotkowitz, M., Van Roy, B., and Lall, S., 2006. "An Approximate Dynamic Programming Approach to Decentralized Control of Stochastic Systems," in Control of Uncertain Systems: Modelling, Approximation, and Design, Springer, Berlin, pp. 243-256.

[CXL19] Chu, Z., Xu, Z., and Li, H., 2019. "New Heuristics for the RCPSP with Multiple Overlapping Modes," Computers and Industrial Engineering, Vol. 131, pp. 146-156.

[CaB07] Camacho, E. F., and Bordons, C., 2007. Model Predictive Control, 2nd Ed., Springer, New York, N. Y.

[Can16] Candy, J. V., 2016. Bayesian Signal Processing: Classical, Modern, and Particle Filtering Methods, Wiley-IEEE Press.

[Cao07] Cao, X. R., 2007. Stochastic Learning and Optimization: A Sensitivity-Based Approach, Springer, N. Y.

[ChC17] Chui, C. K., and Chen, G., 2017. Kalman Filtering, Springer International Publishing.

[ChS00] Christianini, N., and Shawe-Taylor, J., 2000. Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge Univ. Press.

[Che59] Chernoff, H., 1959. "Sequential Design of Experiments," The Annals of Mathematical Statistics, Vol. 30, pp. 755-770.

[Chr97] Christodouleas, J. D., 1997. "Solution Methods for Multiprocessor Network Scheduling Problems with Application to Railroad Operations," Ph.D. Thesis, Operations Research Center, Massachusetts Institute of Technology.

[Cou06] Coulom, R., 2006. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," International Conference on Computers and Games, Springer, pp. 72-83.

[CrS00] Cristianini, N., and Shawe-Taylor, J., 2000. An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge Univ. Press.

[Cyb89] Cybenko, 1989. "Approximation by Superpositions of a Sigmoidal Function," Math. of Control, Signals, and Systems, Vol. 2, pp. 303-314.

[DDF19] Daubechies, I., DeVore, R., Foucart, S., Hanin, B., and Petrova, G., 2019. "Nonlinear Approximation and (Deep) ReLU Networks," arXiv preprint arXiv:1905.02199.

[DFM12] Desai, V. V., Farias, V. F., and Moallemi, C. C., 2012. "Aproximate Dynamic Programming via a Smoothed Approximate Linear Program," Operations Research, Vol. 60, pp. 655-674.

[DFM13] Desai, V. V., Farias, V. F., and Moallemi, C. C., 2013. "Bounds for Markov Decision Processes," in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, by F. Lewis and D. Liu (eds.), IEEE Press, Piscataway, N. J., pp. 452-473.

[DFV03] de Farias, D. P., and Van Roy, B., 2003. "The Linear Programming Approach to Approximate Dynamic Programming," Operations Research, Vol. 51, pp. 850-865.

[DFV04] de Farias, D. P., and Van Roy, B., 2004. "On Constraint Sampling in the Linear Programming Approach to Approximate Dynamic Programming," Mathematics of Operations Research, Vol. 29, pp. 462-478.

[DHS12] Duda, R. O., Hart, P. E., and Stork, D. G., 2012. Pattern Classification, J. Wiley, N. Y.

[DNW16] David, O. E., Netanyahu, N. S., and Wolf, L., 2016. "Deepchess: End-to-End Deep Neural Network for Automatic Learning in Chess," in International Conference on Artificial Neural Networks, pp. 88-96.

[DeA20] Devonport, A., and Arcak, M., 2020. "Estimating Reachable Sets with Scenario Optimization," in Learning for Dynamics and Control, pp. 75-84.

[DeF04] De Farias, D. P., 2004. "The Linear Programming Approach to Approximate Dynamic Programming," in Handbook of Learning and Approximate Dynamic Programming, by J. Si, A. Barto, W. Powell, and D. Wunsch, (Eds.), J. Wiley, N. Y.

[DeK11] Devlin, S., and Kudenko, D., 2011. "Theoretical Considerations of Potential-Based Reward Shaping for Multi-Agent Systems," in Proceedings of AAMAS.

[Den67] Denardo, E. V., 1967. "Contraction Mappings in the Theory Underlying Dynamic Programming," SIAM Review, Vol. 9, pp. 165-177.

[DiL08] Dimitrakakis, C., and Lagoudakis, M. G., 2008. "Rollout Sampling Approximate Policy Iteration," Machine Learning, Vol. 72, pp. 157-171.

[DiM10] Di Castro, D., and Mannor, S., 2010. "Adaptive Bases for Reinforcement Learning," Machine Learning and Knowledge Discovery in Databases, Vol. 6321, pp. 312-327.

[DiW02] Dietterich, T. G., and Wang, X., 2002. "Batch Value Function Approximation via Support Vectors," in Advances in Neural Information Processing Systems, pp. 1491-1498.

[DoJ09] Doucet, A., and Johansen, A. M., 2009. "A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later," Handbook of Nonlinear Filtering, Oxford University Press, Vol. 12, p. 3.

[DrH01] Drezner, Z., and Hamacher, H. W. eds., 2001. Facility Location: Applications and Theory, Springer Science and Business Media.

[DuJ20] Durasevic, M., and Jakobovic, D., 2020. "Automatic Design of Dispatching

Rules for Static Scheduling Conditions," Neural Computing and Applications, Vol. 33, pp. 1-26.

[DuV99] Duin, C., and Voss, S., 1999. "The Pilot Method: A Strategy for Heuristic Repetition with Application to the Steiner Problem in Graphs," Networks: An International Journal, Vol. 34, pp. 181-191.

[EDS18] Efroni, Y., Dalal, G., Scherrer, B., and Mannor, S., 2018. "Beyond the One-Step Greedy Approach in Reinforcement Learning," in Proc. International Conf. on Machine Learning, pp. 1387-1396.

[EMM05] Engel, Y., Mannor, S., and Meir, R., 2005. "Reinforcement Learning with Gaussian Processes," in Proc. of the 22nd ICML, pp. 201-208.

[FHS09] Feitzinger, F., Hylla, T., and Sachs, E. W., 2009. "Inexact Kleinman-Newton Method for Riccati Equations," SIAM J. on Matrix Analysis and Applications, Vol. 3, pp. 272-288.

[FIA03] Findeisen, R., Imsland, L., Allgower, F., and Foss, B.A., 2003. "State and Output Feedback Nonlinear Model Predictive Control: An Overview," European J. of Control, Vol. 9, pp. 190-206.

[FPB15] Farahmand, A. M., Precup, D., Barreto, A. M., and Ghavamzadeh, M., 2015. "Classification-Based Approximate Policy Iteration," IEEE Trans. on Automatic Control, Vol. 60, pp. 2989-2993.

[FeV02] Ferris, M. C., and Voelker, M. M., 2002. "Neuro-Dynamic Programming for Radiation Treatment Planning," Numerical Analysis Group Research Report NA-02/06, Oxford University Computing Laboratory, Oxford University.

[FeV04] Ferris, M. C., and Voelker, M. M., 2004. "Fractionation in Radiation Treatment Planning," Mathematical Programming B, Vol. 102, pp. 387-413.

[Fel60] Feldbaum, A. A., 1960. "Dual Control Theory," Automation and Remote Control, Vol. 21, pp. 874-1039.

[FiV96] Filar, J., and Vrieze, K., 1996. Competitive Markov Decision Processes, Springer.

[FoK09] Forrester, A. I., and Keane, A. J., 2009. "Recent Advances in Surrogate-Based Optimization. Progress in Aerospace Sciences," Vol. 45, pp. 50-79.

[Fra18] Frazier, P. I., 2018. "A Tutorial on Bayesian Optimization," arXiv preprint arXiv:1807.02811.

[Fu17] Fu, M. C., 2017. "Markov Decision Processes, AlphaGo, and Monte Carlo Tree Search: Back to the Future," Leading Developments from INFORMS Communities, INFORMS, pp. 68-88.

[Fun89] Funahashi, K., 1989. "On the Approximate Realization of Continuous Mappings by Neural Networks," Neural Networks, Vol. 2, pp. 183-192.

[GBC16] Goodfellow, I., Bengio, J., and Courville, A., Deep Learning, MIT Press, Cambridge, MA.

[GBL19] Goodson, J. C., Bertazzi, L., and Levary, R. R., 2019. "Robust Dynamic Media Selection with Yield Uncertainty: Max-Min Policies and Dual Bounds," Report.

[GDM19] Guerriero, F., Di Puglia Pugliese, L., and Macrina, G., 2019. "A Rollout Algorithm for the Resource Constrained Elementary Shortest Path Problem," Optimization Methods and Software, Vol. 34, pp. 1056-1074.

[GGS13] Gabillon, V., Ghavamzadeh, M., and Scherrer, B., 2013. "Approximate Dynamic Programming Finally Performs Well in the Game of Tetris," in NIPS, pp. 1754-

1762.

[GGW11] Gittins, J., Glazebrook, K., and Weber, R., 2011. Multi-Armed Bandit Allocation Indices, J. Wiley, N. Y.

[GLG11] Gabillon, V., Lazaric, A., Ghavamzadeh, M., and Scherrer, B., 2011. "Classification-Based Policy Iteration with a Critic," in Proc. of ICML.

[GMP15] Ghavamzadeh, M., Mannor, S., Pineau, J., and Tamar, A., 2015. "Bayesian Reinforcement Learning: A Survey," Foundations and Trends in Machine Learning, Vol. 8, pp. 359-483.

[GSD06] Goodwin, G., Seron, M. M., and De Dona, J. A., 2006. Constrained Control and Estimation: An Optimisation Approach, Springer, N. Y.

[GSS93] Gordon, N. J., Salmond, D. J., and Smith, A. F., 1993. "Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation," in IEE Proceedings, Vol. 140, pp. 107-113.

[GTA17] Gommans, T. M. P., Theunisse, T. A. F., Antunes, D. J., and Heemels, W. P. M. H., 2017. "Resource-Aware MPC for Constrained Linear Systems: Two Rollout Approaches," Journal of Process Control, Vol. 51, pp. 68-83.

[GTO15] Goodson, J. C., Thomas, B. W., and Ohlmann, J. W., 2015. "Restocking-Based Rollout Policies for the Vehicle Routing Problem with Stochastic Demand and Duration Limits," Transportation Science, Vol. 50, pp. 591-607.

[GTO17] Goodson, J. C., Thomas, B. W., and Ohlmann, J. W., 2017. "A Rollout Algorithm Framework for Heuristic Solutions to Finite-Horizon Stochastic Dynamic Programs," European J. of Operational Research, Vol. 258, pp. 216-229.

[GoS84] Goodwin, G. C., and Sin, K. S. S., 1984. Adaptive Filtering, Prediction, and Control, Prentice-Hall, Englewood Cliffs, N. J.

[Gos15] Gosavi, A., 2015. Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning, 2nd Ed., Springer, N. Y.

[Grz17] Grzes, M., 2017. "Reward Shaping in Episodic Reinforcement Learning," in Proc. of the 16th Conference on Autonomous Agents and MultiAgent Systems, pp. 565-573.

[GuM01] Guerriero, F., and Musmanno, R., 2001. "Label Correcting Methods to Solve Multicriteria Shortest Path Problems," J. Optimization Theory Appl., Vol. 111, pp. 589-613.

[GuM03] Guerriero, F., and Mancini, M., 2003. "A Cooperative Parallel Rollout Algorithm for the Sequential Ordering Problem," Parallel Computing, Vol. 29, pp. 663-677.

[Gup20] Gupta, A., 2020. "Existence of Team-Optimal Solutions in Static Teams with Common Information: A Topology of Information Approach," SIAM J. on Control and Optimization, Vol. 58, pp. 998-1021.

[HCR21] Hoffmann, F., Charlish, A., Ritchie, M., and Griffiths, H., 2021. "Policy Rollout Action Selection in Continuous Domains for Sensor Path Planning," IEEE Trans. on Aerospace and Electronic Systems.

[HJG16] Huang, Q., Jia, Q. S., and Guan, X., 2016. "Robust Scheduling of EV Charging Load with Uncertain Wind Power Integration," IEEE Trans. on Smart Grid, Vol. 9, pp. 1043-1054.

[HKT18] Hernandez-Leal, P., Kartal, B., and Taylor, M. E., 2018. "A Survey and Critique of Multiagent Deep Reinforcement Learning," arXiv preprint arXiv:1810.05587.

[HLS06] Han, J., Lai, T. L., and Spivakovsky, V., 2006. "Approximate Policy Optimiza-

tion and Adaptive Control in Regression Models," Computational Economics, Vol. 27, pp. 433-452.

[HLZ19] Ho, T. Y., Liu, S., and Zabinsky, Z. B., 2019. "A Multi-Fidelity Rollout Algorithm for Dynamic Resource Allocation in Population Disease Management," Health Care Management Science, Vol. 22, pp. 727-755.

[HMR19] Hastie, T., Montanari, A., Rosset, S., and Tibshirani, R. J., 2019. "Surprises in High-Dimensional Ridgeless Least Squares Interpolation," arXiv preprint arXiv:1903-.08560.

[HSD17] Hostetler, J., Fern, A., and Dietterich, T., 2017. "Sample-Based Tree Search with Fixed and Adaptive State Abstractions," J. of Artificial Intelligence Research, Vol. 60, pp. 717-777.

[HSS08] Hofmann, T., Scholkopf, B., and Smola, A. J., 2008. "Kernel Methods in Machine Learning," The Annals of Statistics, Vol. 36, pp. 1171-1220.

[HSW89] Hornick, K., Stinchcombe, M., and White, H., 1989. "Multilayer Feedforward Networks are Universal Approximators," Neural Networks, Vol. 2, pp. 359-159.

[HWM19] Hewing, L., Wabersich, K. P., Menner, M., and Zeilinger, M. N., 2019. "Learning-Based Model Predictive Control: Toward Safe Learning in Control," Annual Review of Control, Robotics, and Autonomous Systems.

[HaR21] Hardt, M., and Recht, B., 2021. Patterns, Predictions, and Actions: A Story About Machine Learning, arXiv preprint arXiv:2102.05242.

[HaZ01] Hansen, E. A., and Zilberstein, S., 2001. "LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops," Artificial Intelligence, Vol. 129, pp. 35-62.

[Han98] Hansen, E. A., 1998. "Solving POMDPs by Searching in Policy Space," in Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence, pp. 211-219.

[Hay08] Haykin, S., 2008. Neural Networks and Learning Machines, 3rd Ed., Prentice-Hall, Englewood-Cliffs, N. J.

[HeZ19] Hewing, L., and Zeilinger, M. N., 2019. "Scenario-Based Probabilistic Reachable Sets for Recursively Feasible Stochastic Model Predictive Control," IEEE Control Systems Letters, Vol. 4, pp. 450-455.

[Hew71] Hewer, G., 1971. "An Iterative Technique for the Computation of the Steady State Gains for the Discrete Optimal Regulator," IEEE Trans. on Automatic Control, Vol. 16, pp. 382-384.

[Ho80] Ho, Y. C., 1980. "Team Decision Theory and Information Structures," Proceedings of the IEEE, Vol. 68, pp. 644-654.

[HoF21] Houy, N., and Flaig, J., 2021. "Hospital-Wide Surveillance-Based Antimicrobial Treatments: A Monte-Carlo Look-Ahead Method," Computer Methods and Programs in Biomedicine, Vol. 204, p. 106050.

[HuM16] Huan, X., and Marzouk, Y. M., 2016. "Sequential Bayesian Optimal Experimental Design via Approximate Dynamic Programming," arXiv preprint arXiv:1604.08320.

[Hua15] Huan, X., 2015. Numerical Approaches for Sequential Bayesian Optimal Experimental Design, Ph.D. Thesis, Massachusetts Institute of Technology.

[Hyl11] Hylla, T., 2011. Extension of Inexact Kleinman-Newton Methods to a General Monotonicity Preserving Convergence Theory, Ph.D. Thesis, Univ. of Trier.

[IFT20] Issakkimuthu, M., Fern, A., and Tadepalli, P., 2020. "The Choice Function Framework for Online Policy Improvement," in Proc. of the AAAI Conference on Arti-

ficial Intelligence, Vol. 34, pp. 10178-10185.

[IJT18] Iusem, Jofre, A., and Thompson, P., 2018. "Incremental Constraint Projection Methods for Monotone Stochastic Variational Inequalities," Math. of Operations Research, Vol. 44, pp. 236-263.

[IoS96] Ioannou, P. A., and Sun, J., 1996. Robust Adaptive Control, Prentice-Hall, Englewood Cliffs, N. J.

[JCG20] Jiang, S., Chai, H., Gonzalez, J., and Garnett, R., 2020. "BINOCULARS for Efficient, Nonmyopic Sequential Experimental Design," in Proc. Intern. Conference on Machine Learning, pp. 4794-4803.

[JGJ18] Jones, M., Goldstein, M., Jonathan, P., and Randell, D., 2018. "Bayes Linear Analysis of Risks in Sequential Optimal Design Problems," Electronic J. of Statistics, Vol. 12, pp. 4002-4031.

[JJB20] Jiang, S., Jiang, D. R., Balandat, M., Karrer, B., Gardner, J. R., and Garnett, R., 2020. "Efficient Nonmyopic Bayesian Optimization via One-Shot Multi-Step Trees," arXiv preprint arXiv:2006.15779.

[JiJ17] Jiang, Y., and Jiang, Z. P., 2017. Robust Adaptive Dynamic Programming, J. Wiley, N. Y.

[Jon90] Jones, L. K., 1990. "Constructive Approximations for Neural Networks by Sigmoidal Functions," Proceedings of the IEEE, Vol. 78, pp. 1586-1589.

[JuP07] Jung, T., and Polani, D., 2007. "Kernelizing LSPE(λ)," Proc. 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning, Honolulu, Ha., pp. 338-345.

[KAC15] Kochenderfer, M. J., with Amato, C., Chowdhary, G., How, J. P., Davison Reynolds, H. J., Thornton, J. R., Torres-Carrasquillo, P. A., Ore, N. K., Vian, J., 2015. Decision Making under Uncertainty: Theory and Application, MIT Press, Cambridge, MA.

[KAH15] Khashooei, B. A., Antunes, D. J., and Heemels, W. P. M. H., 2015. "Rollout Strategies for Output-Based Event-Triggered Control," in Proc. 2015 European Control Conference, pp. 2168-2173.

[KBM20] Khosravi, M., Behrunani, V., Myszkorowski, P., Smith, R. S., Rupenyan, A., and Lygeros, J., 2020. "Performance-Driven Cascade Controller Tuning with Bayesian Optimization," arXiv e-prints, pp. arXiv-2007.

[KKK95] Krstic, M., Kanellakopoulos, I., Kokotovic, P., 1995. Nonlinear and Adaptive Control Design, J. Wiley, N. Y.

[KKK20] Kalise, D., Kundu, S., and Kunisch, K., 2020. "Robust Feedback Control of Nonlinear PDEs by Numerical Approximation of High-Dimensional Hamilton-Jacobi-Isaacs Equations." SIAM J. on Applied Dynamical Systems, Vol. 19, pp. 1496-1524.

[KLC98] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R., 1998. "Planning and Acting in Partially Observable Stochastic Domains," Artificial Intelligence, Vol. 101, pp. 99-134.

[KLM82a] Krainak, J. L. S. J. C., Speyer, J., and Marcus, S., 1982. "Static Team Problems - Part I: Sufficient Conditions and the Exponential Cost Criterion," IEEE Trans. on Automatic Control, Vol. 27, pp. 839-848.

[KLM82b] Krainak, J. L. S. J. C., Speyer, J., and Marcus, S., 1982. "Static Team Problems - Part II: Affine Control Laws, Projections, Algorithms, and the LEGT Problem," IEEE Trans. on Automatic Control, Vol. 27, pp. 848-859.

[KMN02] Kearns, M. J., Mansour, Y., and Ng, A. Y., 2002. A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes," Machine Learning, Vol. 49, pp. 193?208.

[KMP06] Keller, P. W., Mannor, S., and Precup, D., 2006. "Automatic Basis Function Construction for Approximate Dynamic Programming and Reinforcement Learning," Proc. of the 23rd ICML, Pittsburgh, Penn.

[KaW94] Kall, P., and Wallace, S. W., 1994. Stochastic Programming, Wiley, Chichester, UK.

[KeG88] Keerthi, S. S., and Gilbert, E. G., 1988. "Optimal, Infinite Horizon Feedback Laws for a General Class of Constrained Discrete Time Systems: Stability and Moving-Horizon Approximations," J. Optimization Theory Appl., Vo. 57, pp. 265-293.

[Kle68] Kleinman, D. L., 1968. "On an Iterative Technique for Riccati Equation Computations," IEEE Trans. Aut. Control, Vol. AC-13, pp. 114-115.

[KoC16] Kouvaritakis, B., and Cannon, M., 2016. Model Predictive Control: Classical, Robust and Stochastic, Springer, N. Y.

[KoG98] Kolmanovsky, I., and Gilbert, E. G., 1998. "Theory and Computation of Disturbance Invariant Sets for Discrete-Time Linear Systems," Math. Problems in Engineering, Vol. 4, pp. 317-367.

[KoS06] Kocsis, L., and Szepesvari, C., 2006. "Bandit Based Monte-Carlo Planning," Proc. of 17th European Conference on Machine Learning, Berlin, pp. 282-293.

[Kok91] Kokotovic, P. V., ed., 1991. Foundations of Adaptive Control, Springer.

[Kre19] Krener, A. J., 2019. "Adaptive Horizon Model Predictive Control and Al'brekht's Method," arXiv preprint arXiv:1904.00053.

[Kri16] Krishnamurthy, V., 2016. Partially Observed Markov Decision Processes, Cambridge Univ. Press.

[KuK21] Kundu, S., and Kunisch, K., 2021. "Policy Iteration for Hamilton-Jacobi-Bellman Equations with Control Constraints," Computational Optimization and Applications, pp. 1-25.

[KuV86] Kumar, P. R., and Varaiya, P. P., 1986. Stochastic Systems: Estimation, Identification, and Adaptive Control, Prentice-Hall, Englewood Cliffs, N. J.

[Kun14] Kung, S. Y., 2014. Kernel Methods and Machine Learning, Cambridge Univ. Press.

[LEC20] Lee, E. H., Eriksson, D., Cheng, B., McCourt, M., and Bindel, D., 2020. "Efficient Rollout Strategies for Bayesian Optimization," arXiv preprint arXiv:2002.10539.

[LGM10] Lazaric, A., Ghavamzadeh, M., and Munos, R., 2010. "Analysis of a Classification-Based Policy Iteration Algorithm," INRIA Report.

[LGW16] Lan, Y., Guan, X., and Wu, J., 2016. "Rollout Strategies for Real-Time Multi-Energy Scheduling in Microgrid with Storage System," IET Generation, Transmission and Distribution, Vol. 10, pp. 688-696.

[LJM19] Li, Y., Johansson, K. H., and Martensson, J., 2019. "Lambda-Policy Iteration with Randomization for Contractive Models with Infinite Policies: Well Posedness and Convergence," arXiv preprint arXiv:1912.08504.

[LKG21] Li, T., Krakow, L. W., and Gopalswamy, S., 2021. "Optimizing Consensus-Based Multi-Target Tracking with Multiagent Rollout Control Policies," arXiv preprint arXiv:2102.02919.

[LLL19] Liu, Z., Lu, J., Liu, Z., Liao, G., Zhang, H. H., and Dong, J., 2019. "Patient Scheduling in Hemodialysis Service," J. of Combinatorial Optimization, Vol. 37, pp. 337-362.

[LLP93] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S., 1993. "Multilayer Feedforward Networks with a Nonpolynomial Activation Function can Approximate any Function," Neural Networks, Vol. 6, pp. 861-867.

[LPS21] Liu, M., Pedrielli, G., Sulc, P., Poppleton, E., and Bertsekas, D. P., 2021. "ExpertRNA: A New Framework for RNA Structure Prediction," bioRxiv 2021.01.18.427087.

[LTZ19] Li, Y., Tang, Y., Zhang, R., and Li, N., 2019. "Distributed Reinforcement Learning for Decentralized Linear Quadratic Control: A Derivative-Free Policy Optimization Approach," arXiv preprint arXiv:1912.09135.

[LWT17] Lowe, L., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I., 2017. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," in Advances in Neural Information Processing Systems, pp. 6379-6390.

[LWW16] Lam, R., Willcox, K., and Wolpert, D. H., 2016. "Bayesian Optimization with a Finite Budget: An Approximate Dynamic Programming Approach," In Advances in Neural Information Processing Systems, pp. 883-891.

[LWW17] Liu, D., Wei, Q., Wang, D., Yang, X., and Li, H., 2017. Adaptive Dynamic Programming with Applications in Optimal Control, Springer, Berlin.

[LZS20] Li, H., Zhang, X., Sun, J., and Dong, X., 2020. "Dynamic Resource Levelling in Projects under Uncertainty," International J. of Production Research.

[LaP03] Lagoudakis, M. G., and Parr, R., 2003. "Reinforcement Learning as Classification: Leveraging Modern Classifiers," in Proc. of ICML, pp. 424-431.

[LaR85] Lai, T., and Robbins, H., 1985. "Asymptotically Efficient Adaptive Allocation Rules," Advances in Applied Math., Vol. 6, pp. 4-22.

[LaW13] Lavretsky, E., and Wise, K., 2013. Robust and Adaptive Control with Aerospace Applications, Springer.

[LaW17] Lam, R., and Willcox, K., 2017. "Lookahead Bayesian Optimization with Inequality Constraints," in Advances in Neural Information Processing Systems, pp. 1890-1900.

[Lee20] Lee, E. H., 2020. "Budget-Constrained Bayesian Optimization, Doctoral dissertation, Cornell University.

[LiL20] Li, Y., and Liu, J., 2020. "Robustly Complete Synthesis of Memoryless Controllers for Nonlinear Systems with Reach-and-Stay Specifications,"IEEE Trans. on Automatic Control.

[LiS16] Liang, S., and Srikant, R., 2016. "Why Deep Neural Networks for Function Approximation?" arXiv preprint arXiv:1610.04161.

[LiW14] Liu, D., and Wei, Q., 2014. "Policy Iteration Adaptive Dynamic Programming Algorithm for Discrete-Time Nonlinear Systems," IEEE Trans. on Neural Networks and Learning Systems, Vol. 25, pp. 621-634.

[LiW15] Li, H., and Womer, N. K., 2015. "Solving Stochastic Resource-Constrained Project Scheduling Problems by Closed-Loop Approximate Dynamic Programming," European J. of Operational Research, Vol. 246, pp. 20-33.

[Lib11] Liberzon, D., 2011. Calculus of Variations and Optimal Control Theory: A Concise Introduction, Princeton Univ. Press.

[MCT10] Mishra, N., Choudhary, A. K., Tiwari, M. K., and Shankar, R., 2010. "Rollout Strategy-Based Probabilistic Causal Model Approach for the Multiple Fault Diagnosis," Robotics and Computer-Integrated Manufacturing, Vol. 26, pp. 325-332.

[MKS15] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and Petersen, S., 2015. "Human-Level Control Through Deep Reinforcement Learning," Nature, Vol. 518, p. 529.

[MLM20] Montenegro, M., Lopez, R., Menchaca-Mendez, R., Becerra, E., and Menchaca-Mendez, R., 2020. "A Parallel Rollout Algorithm for Wildfire Suppression," in Proc. Intern. Congress of Telematics and Computing, pp. 244-255.

[MMB02] McGovern, A., Moss, E., and Barto, A., 2002. "Building a Basic Building Block Scheduler Using Reinforcement Learning and Rollouts," Machine Learning, Vol. 49, pp. 141-160.

[MMS05] Menache, I., Mannor, S., and Shimkin, N., 2005. "Basis Function Adaptation in Temporal Difference Reinforcement Learning," Ann. Oper. Res., Vol. 134, pp. 215-238.

[MPK99] Meuleau, N., Peshkin, L., Kim, K. E., and Kaelbling, L. P., 1999. "Learning Finite-State Controllers for Partially Observable Environments," in Proc. of the 15th Conference on Uncertainty in Artificial Intelligence, pp. 427-436.

[MPP04] Meloni, C., Pacciarelli, D., and Pranzo, M., 2004. "A Rollout Metaheuristic for Job Shop Scheduling Problems," Annals of Operations Research, Vol. 131, pp. 215-235.

[MRR00] Mayne, D., Rawlings, J. B., Rao, C. V., and Scokaert, P. O. M., 2000. "Constrained Model Predictive Control: Stability and Optimality," Automatica, Vol. 36, pp. 789-814.

[MVB20] Magirou, E. F., Vassalos, P., and Barakitis, N., 2020. "A Policy Iteration Algorithm for the American Put Option and Free Boundary Control Problems," J. of Computational and Applied Mathematics, vol. 373, p. 112544.

[MVS19] Muthukumar, V., Vodrahalli, K., and Sahai, A., 2019. "Harmless Interpolation of Noisy Data in Regression," arXiv preprint arXiv:1903.09139.

[MYF03] Moriyama, H., Yamashita, N., and Fukushima, M., 2003. "The Incremental Gauss-Newton Algorithm with Adaptive Stepsize Rule," Computational Optimization and Applications, Vol. 26, pp. 107-141.

[MaJ15] Mastin, A., and Jaillet, P., 2015. "Average-Case Performance of Rollout Algorithms for Knapsack Problems," J. of Optimization Theory and Applications, Vol. 165, pp. 964-984.

[Mac02] Maciejowski, J. M., 2002. Predictive Control with Constraints, Addison-Wesley, Reading, MA.

[Mar55] Marschak, J., 1975. "Elements for a Theory of Teams," Management Science, Vol. 1, pp. 127-137.

[Mar84] Martins, E. Q. V., 1984. "On a Multicriteria Shortest Path Problem," European J. of Operational Research, Vol. 16, pp. 236-245.

[May14] Mayne, D. Q., 2014. "Model Predictive Control: Recent Developments and Future Promise," Automatica, Vol. 50, pp. 2967-2986.

[MeB99] Meuleau, N., and Bourgine, P., 1999. "Exploration of Multi-State Environments: Local Measures and Back-Propagation of Uncertainty," Machine Learning, Vol. 35, pp. 117-154.

[MeK20] Meshram, R., and Kaza, K., 2020. "Simulation Based Algorithms for Markov Decision Processes and Multi-Action Restless Bandits," arXiv preprint arXiv:2007.12933.

[Mey07] Meyn, S., 2007. Control Techniques for Complex Networks, Cambridge Univ. Press, N. Y.

[Min22] Minorsky, N., 1922. "Directional Stability of Automatically Steered Bodies," J. Amer. Soc. Naval Eng., Vol. 34, pp. 280-309.

[MoL99] Morari, M., and Lee, J. H., 1999. "Model Predictive Control: Past, Present, and Future," Computers and Chemical Engineering, Vol. 23, pp. 667-682.

[Mon17] Montgomery, D. C., 2017. Design and Analysis of Experiments, J. Wiley.

[Mun14] Munos, R., 2014. "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning," Foundations and Trends in Machine Learning, Vol. 7, pp. 1-129.

[NHR99] Ng, A. Y., Harada, D., and Russell, S. J., 1999. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping," in Proc. of the 16th International Conference on Machine Learning, pp. 278-287.

[NMS19] Neumann-Brosig, M., Marco, A., Schwarzmann, D., and Trimpe, S., 2019. "Data-Efficient Autotuning with Bayesian Optimization: An Industrial Control Study," IEEE Trans. on Control Systems Technology, Vol. 28, pp. 730-740.

[NMT13] Nayyar, A., Mahajan, A., and Teneketzis, D., 2013. "Decentralized Stochastic Control with Partial History Sharing: A Common Information Approach," IEEE Trans. on Automatic Control, Vol. 58, pp. 1644-1658.

[NNN20] Nguyen, T. T., Nguyen, N. D., and Nahavandi, S., 2020. "Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications," IEEE Trans. on Cybernetics, Vol. 50, pp. 3826-3839.

[NSE19] Nozhati, S., Sarkale, Y., Ellingwood, B., Chong, E. K., and Mahmoud, H., 2019. "Near-Optimal Planning Using Approximate Dynamic Programming to Enhance Post-Hazard Community Resilience Management," Reliability Engineering and System Safety, Vol. 181, pp. 116-126.

[NaA12] Narendra, K. S., and Annaswamy, A. M., 2012. Stable Adaptive Systems, Courier Corporation.

[NaT19] Nayyar, A., and Teneketzis, D., 2019. "Common Knowledge and Sequential Team Problems," IEEE Trans. on Automatic Control, Vol. 64, pp. 5108-5115.

[Ned11] Nedić, A., 2011. "Random Algorithms for Convex Minimization Problems," Math. Programming, Ser. B, Vol. 129, pp. 225-253.

[NgJ13] Ng, A. Y., and Jordan, M. I., 2013. "PEGASUS: A Policy Search Method for Large MDPs and POMDPs," arXiv preprint arXiv:1301.3878.

[Noz21] Nozhati, S., 2021. "A Resilience-Based Framework for Decision Making Based on Simulation-Optimization Approach," Structural Safety, Vol. 89, p. 102032.

[OrH19] OroojlooyJadid, A., and Hajinezhad, D., 2019. "A Review of Cooperative Multi-Agent Deep Reinforcement Learning," arXiv preprint arXiv:1908.03963.

[OrS02] Ormoneit, D., and Sen, S., 2002. "Kernel-Based Reinforcement Learning," Machine Learning, Vol. 49, pp. 161-178.

[PDB92] Pattipati, K. R., Deb, S., Bar-Shalom, Y., and Washburn, R. B., 1992. "A New Relaxation Algorithm and Passive Sensor Data Association," IEEE Trans. Automatic Control, Vol. 37, pp. 198-213.

[PDC14] Pillonetto, G., Dinuzzo, F., Chen, T., De Nicolao, G., and Ljung, L., 2014. "Kernel Methods in System Identification, Machine Learning and Function Estimation: A Survey," Automatica, Vol. 50, pp. 657-682.

[PPB01] Popp, R. L., Pattipati, K. R., and Bar-Shalom, Y., 2001. "m-Best SD Assignment Algorithm with Application to Multitarget Tracking," IEEE Trans. on Aerospace and Electronic Systems, Vol. 37, pp. 22-39.

[PaB99] Patek, S. D., and Bertsekas, D. P., 1999. "Stochastic Shortest Path Games," SIAM J. on Control and Optimization, Vol. 37, pp. 804-824.

[PaR12] Papahristou, N., and Refanidis, I., 2012. "On the Design and Training of Bots to Play Backgammon Variants," in IFIP International Conference on Artificial Intelligence Applications and Innovations, pp. 78-87.

[PaT00] Paschalidis, I. C., and Tsitsiklis, J. N., 2000. "Congestion-Dependent Pricing of Network Services," IEEE/ACM Trans. on Networking, Vol. 8, pp. 171-184.

[PeG04] Peret, L., and Garcia, F., 2004. "On-Line Search for Solving Markov Decision Processes via Heuristic Sampling," in Proc. of the 16th European Conference on Artificial Intelligence, pp. 530-534.

[PeW96] Peng, J., and Williams, R., 1996. "Incremental Multi-Step Q-Learning," Machine Learning, Vol. 22, pp. 283-290.

[PoA69] Pollatschek, M. A., and Avi-Itzhak, B., 1969. "Algorithms for Stochastic Games with Geometrical Interpretation," Management Science, Vol. 15, pp. 399-415.

[PoB04] Poupart, P., and Boutilier, C., 2004. "Bounded Finite State Controllers," in Advances in Neural Information Processing Systems, pp. 823-830.

[PoF08] Powell, W. B., and Frazier, P., 2008. "Optimal Learning," in State-of-the-Art Decision-Making Tools in the Information-Intensive Age, INFORMS, pp. 213-246.

[PoR97] Poore, A. B., and Robertson, A. J. A., 1997. "New Lagrangian Relaxation Based Algorithm for a Class of Multidimensional Assignment Problems," Computational Optimization and Applications, Vol. 8, pp. 129-150.

[PoR12] Powell, W. B., and Ryzhov, I. O., 2012. Optimal Learning, J. Wiley, N. Y.

[Poo94] Poore, A. B., 1994. "Multidimensional Assignment Formulation of Data Association Problems Arising from Multitarget Tracking and Multisensor Data Fusion," Computational Optimization and Applications, Vol. 3, pp. 27-57.

[Pow11] Powell, W. B., 2011. Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Ed., J. Wiley and Sons, Hoboken, N. J.

[Pre95] Prekopa, A., 1995. Stochastic Programming, Kluwer, Boston.

[PuB78] Puterman, M. L., and Brumelle, S. L., 1978. "The Analytic Theory of Policy Iteration," in Dynamic Programming and Its Applications, M. L. Puterman (ed.), Academic Press, N. Y.

[PuB79] Puterman, M. L., and Brumelle, S. L., 1979. "On the Convergence of Policy Iteration in Stationary Dynamic Programming," Mathematics of Operations Research, Vol. 4, pp. 60-69.

[PuS78] Puterman, M. L., and Shin, M. C., 1978. "Modified Policy Iteration Algorithms for Discounted Markov Decision Problems," Management Sci., Vol. 24, pp. 1127-1137.

[PuS82] Puterman, M. L., and Shin, M. C., 1982. "Action Elimination Procedures for Modified Policy Iteration Algorithms," Operations Research, Vol. 30, pp. 301-318.

[Put94] Puterman, M. L., 1994. Markovian Decision Problems, J. Wiley, N. Y.

[QHS05] Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R., and Tucker, P. K., 2005. "Surrogate-Based Analysis and Optimization," Progress in Aerospace Sciences, Vol. 41, pp. 1-28.

[QuL19] Qu, G., and Li, N., "Exploiting Fast Decaying and Locality in Multi-Agent MDP with Tree Dependence Structure," Proc. of 2019 CDC, Nice, France.

[RCR17] Rudi, A., Carratino, L., and Rosasco, L., 2017. "Falkon: An Optimal Large Scale Kernel Method," in Advances in Neural Information Processing Systems, pp. 3888-3898.

[RGG21] Rimélé, A., Grangier, P., Gamache, M., Gendreau, M., and Rousseau, L. M., 2021. "E-Commerce Warehousing: Learning a Storage Policy," arXiv:2101.08828.

[RKL21] Rupenyan, A., Khosravi, M., and Lygeros, J., 2021. "Performance-Based Trajectory Optimization for Path Following Control Using Bayesian Optimization," arXiv preprint arXiv:2103.15416.

[RMD17] Rawlings, J. B., Mayne, D. Q., and Diehl, M. M., 2017. Model Predictive Control: Theory, Computation, and Design, 2nd Ed., Nob Hill Publishing (updated in 2019 and 2020).

[RPF12] Ryzhov, I. O., Powell, W. B., and Frazier, P. I., 2012. "The Knowledge Gradient Algorithm for a General Class of Online Learning Problems," Operations Research, Vol. 60, pp. 180-195.

[RPP08] Ross, S., Pineau, J., Paquet, S., and Chaib-Draa, B., 2008. "Online Planning Algorithms for POMDPs," J. of Artificial Intelligence Research, Vol. 32, pp. 663-704.

[RSM08] Reisinger, J., Stone, P., and Miikkulainen, R., 2008. "Online Kernel Selection for Bayesian Reinforcement Learning," in Proc. of the 25th International Conference on Machine Learning, pp. 816-823.

[RaF91] Raghavan, T. E. S., and Filar, J. A., 1991. "Algorithms for Stochastic Games - A Survey," Zeitschrift fur Operations Research, Vol. 35, pp. 437-472.

[RaR17] Rawlings, J. B., and Risbeck, M. J., 2017. "Model Predictive Control with Discrete Actuators: Theory and Application," Automatica, Vol. 78, pp. 258-265.

[RaW06] Rasmussen, C. E., and Williams, C. K., 2006. Gaussian Processes for Machine Learning, MIT Press, Cambridge, MA.

[Rad62] Radner, R., 1962. "Team Decision Problems," Ann. Math. Statist., Vol. 33, pp. 857-881.

[Rek64] Rekasius, Z. V., 1964. "Suboptimal Design of Intentionally Nonlinear Controllers," IEEE Trans. on Automatic Control, Vol. 9, pp. 380-386.

[RoB17] Rosolia, U., and Borrelli, F., 2017. "Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework," IEEE Trans. on Automatic Control, Vol. 63, pp. 1883-1896.

[RoB19] Rosolia, U., and Borrelli, F., 2019. "Sample-Based Learning Model Predictive Control for Linear Uncertain Systems," 58th Conference on Decision and Control (CDC), pp. 2702-2707.

[Rob52] Robbins, H., 1952. "Some Aspects of the Sequential Design of Experiments," Bulletin of the American Mathematical Society, Vol. 58, pp. 527-535.

[Ros70] Ross, S. M., 1970. Applied Probability Models with Optimization Applications, Holden-Day, San Francisco, CA.

[Ros12] Ross, S. M., 2012. Simulation, 5th Ed., Academic Press, Orlando, Fla.

[Rot79] Rothblum, U. G., 1979. "Iterated Successive Approximation for Sequential Decision Processes," in Stochastic Control and Optimization, by J. W. B. van Overhagen and H. C. Tijms (eds), Vrije University, Amsterdam.

[RuK16] Rubinstein, R. Y., and Kroese, D. P., 2016. Simulation and the Monte Carlo Method, 3rd Ed., J. Wiley, N. Y.

[RuN94] Rummery, G. A., and Niranjan, M., 1994. "On-Line Q-Learning Using Connectionist Systems," University of Cambridge, England, Department of Engineering, TR-166.

[RuN16] Russell, S. J., and Norvig, P., 2016. Artificial Intelligence: A Modern Approach, Pearson Education Limited, Malaysia.

[RuS03] Ruszczynski, A., and Shapiro, A., 2003. "Stochastic Programming Models," in Handbooks in Operations Research and Management Science, Vol. 10, pp. 1-64.

[SDY20] Schope, M. I., Driessen, H., and Yarovoy, A., 2020. "Multi-Task Sensor Resource Balancing Using Lagrangian Relaxation and Policy Rollout," in 2020 IEEE 23rd International Conference on Information Fusion (FUSION), pp. 1-8.

[SGC02] Savagaonkar, U., Givan, R., and Chong, E. K. P., 2002. "Sampling Techniques for Zero-Sum, Discounted Markov Games," in Proc. 40th Allerton Conference on Communication, Control and Computing, Monticello, Ill.

[SGG15] Scherrer, B., Ghavamzadeh, M., Gabillon, V., Lesner, B., and Geist, M., 2015. "Approximate Modified Policy Iteration and its Application to the Game of Tetris," J. of Machine Learning Research, Vol. 16, pp. 1629-1676.

[SHB15] Simroth, A., Holfeld, D., and Brunsch, R., 2015. "Job Shop Production Planning under Uncertainty: A Monte Carlo Rollout Approach," Proc. of the International Scientific and Practical Conference, Vol. 3, pp. 175-179.

[SHM16] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., and Dieleman, S., 2016. "Mastering the Game of Go with Deep Neural Networks and Tree Search," Nature, Vol. 529, pp. 484-489.

[SHS17] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., and Lillicrap, T., 2017. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," arXiv preprint arXiv:1712.01815.

[SJL18] Soltanolkotabi, M., Javanmard, A., and Lee, J. D., 2018. "Theoretical Insights into the Optimization Landscape of Over-Parameterized Shallow Neural Networks," IEEE Trans. on Information Theory, Vol. 65, pp. 742-769.

[SLA12] Snoek, J., Larochelle, H., and Adams, R. P., 2012. "Practical Bayesian Optimization of Machine Learning Algorithms," in Advances in Neural Information Processing Systems, pp. 2951-2959.

[SLJ13] Sun, B., Luh, P. B., Jia, Q. S., Jiang, Z., Wang, F., and Song, C., 2013. "Building Energy Management: Integrated Control of Active and Passive Heating, Cooling, Lighting, Shading, and Ventilation Systems," IEEE Trans. on Automation Science and Engineering, Vol. 10, pp. 588-602.

[SNC18] Sarkale, Y., Nozhati, S., Chong, E. K., Ellingwood, B. R., and Mahmoud, H., 2018. "Solving Markov Decision Processes for Network-Level Post-Hazard Recovery via Simulation Optimization and Rollout," in 2018 IEEE 14th International Conference on

Automation Science and Engineering, pp. 906-912.

[SSS17] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., and Chen, Y., 2017. "Mastering the Game of Go Without Human Knowledge," Nature, Vol. 550, pp. 354-359.

[SSW16] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N., 2015. "Taking the Human Out of the Loop: A Review of Bayesian Optimization," Proc. of IEEE, Vol. 104, pp. 148-175.

[SVR10] Stewart, B. T., Venkat, A. N., Rawlings, J. B., Wright, S. J., and Pannocchia, G., 2010. "Cooperative Distributed Model Predictive Control," Systems and Control Letters, Vol. 59, pp. 460-469.

[SWM89] Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., 1989. "Design and Analysis of Computer Experiments," Statistical Science, Vol. 4, pp. 409-423.

[SWR11] Stewart, B. T., Wright, S. J., and Rawlings, J. B., 2011. "Cooperative Distributed Model Predictive Control for Nonlinear Systems," Journal of Process Control, Vol. 21, pp. 698-704.

[SYL17] Saldi, N., Yuksel, S., and Linder, T., 2017. "Finite Model Approximations for Partially Observed Markov Decision Processes with Discounted Cost," arXiv preprint arXiv:1710.07009.

[SZL08] Sun, T., Zhao, Q., Lun, P., and Tomastik, R., 2008. "Optimization of Joint Replacement Policies for Multipart Systems by a Rollout Framework," IEEE Trans. on Automation Science and Engineering, Vol. 5, pp. 609-619.

[SaB11] Sastry, S., and Bodson, M., 2011. Adaptive Control: Stability, Convergence and Robustness, Courier Corporation.

[SaL79] Saridis, G. N., and Lee, C.-S. G., 1979. "An Approximation Theory of Optimal Control for Trainable Manipulators," IEEE Trans. Syst., Man, Cybernetics, Vol. 9, pp. 152-159.

[SaR04] Santos, M. S., and Rust, J., 2004. "Convergence Properties of Policy Iteration," SIAM J. on Control and Optimization, Vol. 42, pp. 2094-2115.

[Sal21] Saldi, N., 2021. "Regularized Stochastic Team Problems," Systems and Control Letters, Vol. 149.

[Sas02] Sasena, M. J., 2002. Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations, Ph.D. Thesis, Univ. of Michigan.

[ScS02] Scholkopf, B., and Smola, A. J., 2002. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, Cambridge, MA.

[Sch13] Scherrer, B., 2013. "Performance Bounds for Lambda Policy Iteration and Application to the Game of Tetris," J. of Machine Learning Research, Vol. 14, pp. 1181-1227.

[Sco10] Scott, S. L., 2010. "A Modern Bayesian Look at the Multi-Armed Bandit," Applied Stochastic Models in Business and Industry, Vol. 26, pp. 639-658.

[Sec00] Secomandi, N., 2000. "Comparing Neuro-Dynamic Programming Algorithms for the Vehicle Routing Problem with Stochastic Demands," Computers and Operations Research, Vol. 27, pp. 1201-1225.

[Sec01] Secomandi, N., 2001. "A Rollout Policy for the Vehicle Routing Problem with Stochastic Demands," Operations Research, Vol. 49, pp. 796-802.

[Sec03] Secomandi, N., 2003. "Analysis of a Rollout Approach to Sequencing Problems with Stochastic Routing Applications," J. of Heuristics, Vol. 9, pp. 321-352.

[ShC04] Shawe-Taylor, J., and Cristianini, N., 2004. Kernel Methods for Pattern Analysis, Cambridge Univ. Press.

[Sha50] Shannon, C., 1950. "Programming a Digital Computer for Playing Chess," Phil. Mag., Vol. 41, pp. 356-375.

[Sha53] Shapley, L. S., 1953. "Stochastic Games," Proc. of the National Academy of Sciences, Vol. 39, pp. 1095-1100.

[SiK19] Singh, R., and Kumar, P. R., 2019. "Optimal Decentralized Dynamic Policies for Video Streaming over Wireless Channels," arXiv preprint arXiv:1902.07418.

[SiV10] Silver, D., and Veness, J., 2010. "Monte-Carlo Planning in Large POMDPs," in Proc. 23rd Int. Conf. NeurIPS, pp. 2164-2172.

[SlL91] Slotine, J.-J. E., and Li, W., Applied Nonlinear Control, Prentice-Hall, Englewood Cliffs, N. J.

[StW91] Stewart, B. S., and White, C. C., 1991. "Multiobjective A^*," J. ACM, Vol. 38, pp. 775-814.

[SuB18] Sutton, R., and Barto, A. G., 2018. Reinforcement Learning, 2nd Ed., MIT Press, Cambridge, MA.

[SuY19] Su, L., and Yang, P., 2019. 'On Learning Over-Parameterized Neural Networks: A Functional Approximation Perspective," in Advances in Neural Information Processing Systems, pp. 2637-2646.

[Sun19] Sun, R., 2019. "Optimization for Deep Learning: Theory and Algorithms," arXiv preprint arXiv:1912.08957.

[Sze10] Szepesvari, C., 2010. Algorithms for Reinforcement Learning, Morgan and Claypool Publishers, San Franscisco, CA.

[TBP21] Tuncel, Y., Bhat, G., Park, J., and Ogras, U., 2021. "ECO: Enabling Energy-Neutral IoT Devices through Runtime Allocation of Harvested Energy," arXiv preprint arXiv:2102.13605.

[TCW19] Tseng, W. J., Chen, J. C., Wu, I. C., and Wei, T. H., 2019. "Comparison Training for Computer Chinese Chess," IEEE Trans. on Games, Vol. 12, pp. 169-176.

[TGL13] Tesauro, G., Gondek, D. C., Lenchner, J., Fan, J., and Prager, J. M., 2013. "Analysis of Watson's Strategies for Playing Jeopardy!," J. of Artificial Intelligence Research, Vol. 47, pp. 205-251.

[TRV16] Tu, S., Roelofs, R., Venkataraman, S., and Recht, B., 2016. "Large Scale Kernel Learning Using Block Coordinate Descent," arXiv preprint arXiv:1602.05310.

[TaL20] Tanzanakis, A., and Lygeros, J., 2020. "Data-Driven Control of Unknown Systems: A Linear Programming Approach," arXiv preprint arXiv:2003.00779.

[TeG96] Tesauro, G., and Galperin, G. R., 1996. "On-Line Policy Improvement Using Monte Carlo Search," NIPS, Denver, CO.

[Tes89a] Tesauro, G. J., 1989. "Neurogammon Wins Computer Olympiad," Neural Computation, Vol. 1, pp. 321-323.

[Tes89b] Tesauro, G. J., 1989. "Connectionist Learning of Expert Preferences by Comparison Training," in Advances in Neural Information Processing Systems, pp. 99-106.

[Tes92] Tesauro, G. J., 1992. "Practical Issues in Temporal Difference Learning," Machine Learning, Vol. 8, pp. 257-277.

[Tes94] Tesauro, G. J., 1994. "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play," Neural Computation, Vol. 6, pp. 215-219.

[Tes95] Tesauro, G. J., 1995. "Temporal Difference Learning and TD-Gammon," Communications of the ACM, Vol. 38, pp. 58-68.

[Tes01] Tesauro, G. J., 2001. "Comparison Training of Chess Evaluation Functions," in Machines that Learn to Play Games, Nova Science Publishers, pp. 117-130.

[Tes02] Tesauro, G. J., 2002. "Programming Backgammon Using Self-Teaching Neural Nets," Artificial Intelligence, Vol. 134, pp. 181-199.

[ThS09] Thiery, C., and Scherrer, B., 2009. "Improvements on Learning Tetris with Cross-Entropy," International Computer Games Association J., Vol. 32, pp. 23-33.

[TsV96] Tsitsiklis, J. N., and Van Roy, B., 1996. "Feature-Based Methods for Large-Scale Dynamic Programming," Machine Learning, Vol. 22, pp. 59-94.

[TsV97] Tsitsiklis, J. N., and Van Roy, B., 1997. "An Analysis of Temporal-Difference Learning with Function Approximation," IEEE Trans. on Aut. Control, Vol. 42, pp. 674-690.

[TsV99] Tsitsiklis, J. N., and Van Roy, B., 1999. "Optimal Stopping of Markov Processes: Hilbert Space Theory, Approximation Algorithms, and an Application to Pricing Financial Derivatives", IEEE Trans. on Aut. Control, Vol. 44, pp. 1840-1851.

[Tse98] Tseng, P., 1998. "Incremental Gradient(-Projection) Method with Momentum Term and Adaptive Stepsize Rule," SIAM J. on Optimization, Vol. 8, pp. 506-531.

[Tsi94] Tsitsiklis, J. N., 1994. "Asynchronous Stochastic Approximation and Q-Learning," Machine Learning, Vol. 16, pp. 185-202.

[TuP03] Tu, F., and Pattipati, K. R., 2003. "Rollout Strategies for Sequential Fault Diagnosis," IEEE Trans. on Systems, Man and Cybernetics, Part A, pp. 86-99.

[UGM18] Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., and Hennig, M., 2018. "Offline-Online Approximate Dynamic Programming for Dynamic Vehicle Routing with Stochastic Requests," Transportation Science, Vol. 53, pp. 185-202.

[Ulm17] Ulmer, M. W., 2017. Approximate Dynamic Programming for Dynamic Vehicle Routing, Springer, Berlin.

[VBC19] Vinyals, O., Babuschkin, I., Czarnecki, W. M., and thirty nine more authors, 2019. "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning," Nature, Vol. 575, p. 350.

[VPA09] Vrabie, D., Pastravanu, O., Abu-Khalaf, M., and Lewis, F. L., 2009. "Adaptive Optimal Control for Continuous-Time Linear Systems Based on Policy Iteration," Automatica, Vol. 45, pp. 477-484.

[VVL13] Vrabie, D., Vamvoudakis, K. G., and Lewis, F. L., 2013. Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles, The Institution of Engineering and Technology, London.

[Van76] Van Nunen, J. A., 1976. Contracting Markov Decision Processes, Mathematical Centre Report, Amsterdam.

[WCG02] Wu, G., Chong, E. K. P., and Givan, R. L., 2002. "Burst-Level Congestion Control Using Hindsight Optimization," IEEE Trans. on Aut. Control, Vol. 47, pp. 979-991.

[WCG03] Wu, G., Chong, E. K. P., and Givan, R. L., 2003. "Congestion Control Using Policy Rollout," Proc. 2nd IEEE CDC, Maui, Hawaii, pp. 4825-4830.

[WOB15] Wang, Y., O'Donoghue, B., and Boyd, S., 2015. "Approximate Dynamic Programming via Iterated Bellman Inequalities," International J. of Robust and Nonlinear Control, Vol. 25, pp. 1472-1496.

[WaB15] Wang, M., and Bertsekas, D. P., 2015. "Incremental Constraint Projection Methods for Variational Inequalities," Mathematical Programming, Vol. 150, pp. 321-363.

[WaB16] Wang, M., and Bertsekas, D. P., 2016. "Stochastic First-Order Methods with Random Constraint Projection," SIAM J. on Optimization, Vol. 26, pp. 681-717.

[WaS00] de Waal, P. R., and van Schuppen, J. H., 2000. "A Class of Team Problems with Discrete Action Spaces: Optimality Conditions Based on Multimodularity," SIAM J. on Control and Optimization, Vol. 38, pp. 875-892.

[Wat89] Watkins, C. J. C. H., Learning from Delayed Rewards, Ph.D. Thesis, Cambridge Univ., England.

[WeB99] Weaver, L., and Baxter, J., 1999. "Learning from State Differences: STD(λ)," Tech. Report, Dept. of Computer Science, Australian National University.

[WhS94] White, C. C., and Scherer, W. T., 1994. "Finite-Memory Suboptimal Design for Partially Observed Markov Decision Processes," Operations Research, Vol. 42, pp. 439-455.

[Whi88] Whittle, P., 1988. "Restless Bandits: Activity Allocation in a Changing World," J. of Applied Probability, pp. 287-298.

[Whi91] White, C. C., 1991. "A Survey of Solution Techniques for the Partially Observed Markov Decision Process," Annals of Operations Research, Vol. 32, pp. 215-230.

[WiB93] Williams, R. J., and Baird, L. C., 1993. "Analysis of Some Incremental Variants of Policy Iteration: First Steps Toward Understanding Actor-Critic Learning Systems," Report NU-CCS-93-11, College of Computer Science, Northeastern University, Boston, MA.

[WiS98] Wiering, M., and Schmidhuber, J., 1998. "Fast Online Q(λ)," Machine Learning, Vol. 33, pp. 105-115.

[Wie03] Wiewiora, E., 2003. "Potential-Based Shaping and Q-Value Initialization are Equivalent," J. of Artificial Intelligence Research, Vol. 19, pp. 205-208.

[Wit66] Witsenhausen, H. S., 1966. Minimax Control of Uncertain Systems, Ph.D. thesis, MIT.

[Wit68] Witsenhausen, H., 1968. "A Counterexample in Stochastic Optimum Control," SIAM J. on Control, Vol. 6, pp. 131-147.

[Wit71a] Witsenhausen, H. S., 1971. "On Information Structures, Feedback and Causality," SIAM J. Control, Vol. 9, pp. 149-160.

[Wit71b] Witsenhausen, H., 1971. "Separation of Estimation and Control for Discrete Time Systems," Proceedings of the IEEE, Vol. 59, pp. 1557-1566.

[XLX21] Xie, F., Li, H., and Xu, Z., 2021. "An Approximate Dynamic Programming Approach to Project Scheduling with Uncertain Resource Availabilities," Applied Mathematical Modelling.

[YDR04] Yan, X., Diaconis, P., Rusmevichientong, P., and Van Roy, B., 2004. "Solitaire: Man Versus Machine," Advances in Neural Information Processing Systems, Vol. 17, pp.

1553-1560.

[YSH17] Ye, N., Somani, A., Hsu, D., and Lee, W.S., 2017. "DESPOT: Online POMDP Planning with Regularization," J. of Artificial Intelligence Research, Vol. 58, pp. 231-266.

[YWX20] Yan, S., Wang, X., and Xu, L., 2020. "Rollout Algorithm for Light-Weight Physical-Layer Authentication in Cognitive Radio Networks," IET Communications, Vol. 14, pp. 3128-3134.

[YYM19] Yu, L., Yang, H., Miao, L., and Zhang, C., 2019. "Rollout Algorithms for Resource Allocation in Humanitarian Logistics," IISE Trans., Vol. 51, pp. 887-909.

[Yar17] Yarotsky, D., 2017. "Error Bounds for Approximations with Deep ReLU Networks," Neural Networks, Vol. 94, pp. 103-114.

[YuB07] Yu, H., and Bertsekas, D. P., 2007. "A Least Squares Q-Learning Algorithm for Optimal Stopping Problems," Proc. European Control Conference 2007, Kos, Greece, pp. 2368-2375; an extended version appears in Lab. for Information and Decision Systems Report LIDS-P-2731, MIT.

[YuB08] Yu, H., and Bertsekas, D. P., 2008. "On Near-Optimality of the Set of Finite-State Controllers for Average Cost POMDP," Math. of OR, Vol. 33, pp. 1-11.

[YuB09] Yu, H., and Bertsekas, D. P., 2009. "Basis Function Adaptation Methods for Cost Approximation in MDP," Proceedings of 2009 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2009), Nashville, Tenn.

[YuB13a] Yu, H., and Bertsekas, D. P., 2013. "Q-Learning and Policy Iteration Algorithms for Stochastic Shortest Path Problems," Annals of Operations Research, Vol. 208, pp. 95-132.

[YuB13b] Yu, H., and Bertsekas, D. P., 2013. "On Boundedness of Q-Learning Iterates for Stochastic Shortest Path Problems," Math. of OR, Vol. 38, pp. 209-227.

[YuB15] Yu, H., and Bertsekas, D. P., 2015. "A Mixed Value and Policy Iteration Method for Stochastic Control with Universally Measurable Policies," Math. of OR, Vol. 40, pp. 926-968.

[YuK20] Yue, X., and Kontar, R. A., 2020. "Lookahead Bayesian Optimization via Rollout: Guarantees and Sequential Rolling Horizons," arXiv preprint arXiv:1911.01004.

[Yu14] Yu, H., 2014. "Stochastic Shortest Path Games and Q-Learning," arXiv preprint arXiv:1412.8570.

[Yu20] Yu, H., 2020. "Average Cost Optimality Inequality for Markov Decision Processes with Borel Spaces and Universally Measurable Policies," SIAM J. on Control and Optimization, Vol. 58, pp. 2469-2502.

[Yu21] Yu, H., 2021. "Average-Cost Optimality Results for Borel-Space Markov Decision Processes with Universally Measurable Policies," arXiv preprint arXiv:2104.00181.

[Yua19] Yuanhong, L. I. U., 2019. "Optimal Selection of Tests for Fault Detection and Isolation in Multi-Operating Mode System," Journal of Systems Engineering and Electronics, Vol. 30, pp. 425-434.

[ZBH16] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O., 2016. "Understanding Deep Learning Requires Rethinking Generalization," arXiv preprint arXiv: 1611.03530.

[ZBH21] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O., 2021. "Understanding Deep Learning (Still) Requires Rethinking Generalization," Communications of the ACM, VOL. 64, pp. 107-115.

[ZKY20] Zhang, H., Kafouros, M., and Yu, Y., 2020. "PlanLight: Learning to Optimize Traffic Signal Control With Planning and Iterative Policy Improvement," IEEE Access, Vol. 8, pp. 219244-219255.

[ZOT18] Zhang, S., Ohlmann, J. W., and Thomas, B. W., 2018. "Dynamic Orienteering on a Network of Queues," Transportation Science, Vol. 52, pp. 691-706.

[ZSG20] Zoppoli, R., Sanguineti, M., Gnecco, G., and Parisini, T., 2020. Neural Approximations for Optimal Control and Decision, Springer.

[ZYB19] Zhang, K., Yang, Z., and Basar, T., 2019. "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," arXiv preprint arXiv:1911.10635.

[ZuS81] Zuker, M., and Stiegler, P., 1981. "Optimal Computer Folding of Larger RNA Sequences Using Thermodynamics and Auxiliary Information," Nucleic Acids Res., Vol. 9, pp. 133-148.