A Course in Reinforcement Learning (2nd Edition)

by

Dimitri P. Bertsekas

Arizona State University

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

Publisher's Cataloging-in-Publication Data

Bertsekas, Dimitri P.
A Course in Reinforcement Learning (2nd Edition)
Includes Bibliography and Index
1. Mathematical Optimization. 2. Dynamic Programming. I. Title.
QA402.5 .B465 2020 519.703 00-91281

ISBN-10: 1-886529-29-9, ISBN-13: 978-1-886529-29-8

Latest editorial and other changes added on 1/14/2025

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. From 1979 to 2019 he was a professor at the Electrical Engineering and Computer Science Department of the Massachusetts Institute of Technology (M.I.T.), where he continues to hold the title of McAfee Professor of Engineering. In 2019, he joined the School of Computing and Augmented Intelligence 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 twenty 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." At ASU, he has been focusing in teaching and research in reinforcement learning, and he has written several textbooks and research monographs in this field since 2019.

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."

CONTENTS

2. Approximation in Value Space - Rollout Algorithms

Preface

It is a capital mistake to theorize before one has data.

Sherlock Holmes

We are surrounded by data, but starved for insights.

Jay Baer

This book is based on lecture notes for the 2024 ASU research-oriented course on Reinforcement Learning (RL) that I have offered in each of the last six years, as the field was rapidly evolving.† The purpose of the book is to give an overview of the RL methodology, with a particular focus on problems of optimal and suboptimal control, as well as discrete optimization. More broadly, the book aims to provide a framework for structured thinking about RL and its connections to decision and control, which is grounded in mathematics, but is not dominated by it.

Generally, RL can be viewed as the art and science of sequential decision making for large and difficult problems, often in the presence of imprecisely known and changing environment conditions.‡ Dynamic Programming (DP), a well-established optimization methodology, forms the

† The 1st edition of the book appeared in 2023. The 2nd edition includes material that was introduced in the 2024 offering of the ASU course.

‡ The term "reinforcement learning" originated from the concept of "reinforcement" in behavioral psychology. Its meaning has evolved over time, starting in the 1980s.

theoretical foundation of RL. This is unlikely to change in the future, despite the rapid pace of technological innovation. In fact, there are strong connections between sequential decision making and the new wave of technological change, generative artificial intelligence, transformers, GPT applications, and natural language processing, as we will aim to show in this book.

In DP there are two principal objects to compute: the *optimal value function*, which provides the optimal cost that can be attained starting from any given initial state, and the *optimal policy*, which provides the optimal decision to apply at any given state and time. Unfortunately, the exact application of DP runs into formidable computational challenges, commonly known as the *curse of dimensionality*. To address these, RL aims to approximate the optimal value function and policy, by using manageable off-line and/or on-line computation, which often involves neural networks (hence the alternative name Neuro-Dynamic Programming [BeT96]).

Thus there are two major methodological approaches in RL: *approximation in value space*, where we approximate in some way the optimal value function, and *approximation in policy space*, where we construct a suboptimal policy by optimizing over a restricted set of policies. These two approaches can sometimes be combined to exploit the strengths of both. Generally, approximation in value space aligns more closely to the central DP ideas of value and policy iteration, while approximation in policy space often relies on gradient-like descent, a more broadly applicable optimization methodology.

The book focuses primarily on approximation in value space, with limited coverage of approximation in policy space in Chapter 3. However, it is structured to allow instructors to go into approximation in policy space in greater detail, using any of a number of available sources.

Our Conceptual Framework

An important part of our line of development is a new conceptual framework, which aims to bridge the gaps between the artificial intelligence, control theory, and operations research views of our subject. This framework, the focus of the author's recent monograph "Lessons from AlphaZero ...", [Ber22a], centers on approximate forms of DP that are inspired by some of the major successes of RL involving games. Primary examples are the recent (2017) AlphaZero program (which plays chess), and the similarly structured and earlier (1990s) TD-Gammon program (which plays backgammon).

Our framework is couched on two general algorithms that are designed largely independently of each other and operate in synergy through the powerful mechanism of Newton's method, applied to the fundamental Bellman equation of DP. We call these the *off-line training* and the *on-line play* algorithms. In the AlphaZero and TD-Gammon game contexts, the

off-line training algorithm is the method used to teach the program how to evaluate positions and to generate good moves at any given position, while the on-line play algorithm is the method used to play in real time against human or computer opponents.

Our synergistic view of off-line training and on-line play is motivated by some striking empirical observations. In particular, both AlphaZero and TD-Gammon were trained off-line extensively using neural networks and an approximate version of the fundamental DP algorithm of policy iteration. Yet the AlphaZero player that was obtained off-line is not used directly during on-line play (it is too inaccurate due to approximation errors that are inherent in off-line neural network training). Instead, a separate on-line player is used to select moves, based on multistep lookahead minimization and a terminal position evaluator that was trained using experience with the off-line player. The on-line player performs a form of policy improvement, which is not degraded by neural network approximations. As a result, it greatly improves the performance of the off-line player.

Similarly, TD-Gammon performs on-line a policy improvement step using one-step or two-step lookahead minimization, which is not degraded by neural network approximations. To this end, it uses an off-line neural network-trained terminal position evaluator, and importantly it also extends its on-line lookahead by rollout (simulation with the one-step lookahead player that is based on the position evaluator). Thus in summary:

(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 exact policy improvement with long lookahead minimization, which corrects for the inevitable imperfections of the neural network-trained off-line player, and position evaluator/terminal cost approximation.

(b) The TD-Gammon player that uses long rollout plays much better than TD-Gammon without rollout. This is due to the beneficial effect of the rollout, which serves as a supplement and substitute for long lookahead minimization.

An important lesson from AlphaZero and TD-Gammon is that the performance of an off-line trained policy can be greatly improved by on-line approximation in value space, with long lookahead (involving minimization or rollout with the off-line policy, or both), and terminal cost approximation that is obtained off-line. This performance enhancement is often dramatic and is due to a simple fact, which is couched on algorithmic mathematics and is a focal point of our course: *approximation in value space with one-step lookahead minimization amounts to a step of Newton's method for solving Bellman's equation, while the starting point for the Newton step is based on the results of off-line training, and may be enhanced by longer lookahead minimization and on-line rollout*. Indeed the major determinant of the quality of the on-line policy is the Newton step that is performed

on-line, while off-line training plays a secondary role by comparison.

An additional benefit of policy improvement by approximation in value space, not observed in the context of games (which have stable rules and environment), is that it works well with changing problem parameters and on-line replanning, similar to the methodology of indirect adaptive control. In particular, the Bellman equation is perturbed due to the parameter changes, but approximation in value space still operates as a Newton step. An essential requirement here is that a system model is estimated on-line through some identification method, and is used during the one-step or multistep lookahead minimization process. This may be a mathematical model represented by equations and probability distributions, or a computer model that involves simulation and/or neural network software.

In this book, we will describe the basic RL methodologies, and we will aim to explain (often with visualization) their effectiveness (or lack thereof) through the lens of the off-line and on-line synergy. In the process, we will bring out the strong connections between the artificial intelligence view of RL, the control theory view of sequential decision making, and the operations research view of discrete optimization. Moreover, we will describe a broad variety of algorithms that can be used for on-line play.

In particular, we will aim to show that the methodology of approximation in value space and rollout applies very broadly to deterministic and stochastic optimal control problems, involving both discrete and continuous search spaces, as well as finite and infinite horizon. We will also show that our conceptual framework can be effectively integrated with other important methodologies such as multiagent systems and decentralized control, discrete and Bayesian optimization, two-person antagonistic games, and heuristic algorithms for discrete optimization.

Reinforcement Learning and Model Predictive Control

Significantly, the synergy between off-line training and on-line play also underlies Model Predictive Control (MPC), a major control system design methodology that has been extensively developed since the 1980s.† One of the major themes of this book is that RL, as practiced by the artificial intelligence community, and MPC, as practiced by the decision and control community, *can be viewed as essentially identical methodologies at a mathematical and conceptual level*.

The similarity of the two fields will become evident from the discussion of Section 1.1. In particular, both fields are founded on DP, and

† "Model predictive control" is an abbreviation for the far more descriptive term "model-based predictive control." Here, the word "model" refers to either a mathematical model or a computer software model of a discrete-time dynamic system that evolves under the influence of control. The term "predictive" generally refers to taking into account the system's future, while applying control in the present.

both involve an on-line search algorithm and an off-line training process. Their differences are largely a matter of emphasis: RL has its roots on theories of game playing and learning, and places far more emphasis in off-line training, and discrete state and action spaces, while MPC has its roots in predictive on-line optimization and places far more emphasis in on-line play, and continuous state and control spaces. Other differences, which can also be traced to the respective origins of the two fields, are that RL includes a focus on approximation in policy space (which has been of peripheral interest in MPC), while MPC includes a strong focus on closed-loop system stability issues (which are hardly ever discussed in the context of RL).

Supporting Literature

In this book, we will deemphasize mathematical proofs, and focus instead on visualizations and intuitive (but still rigorous) explanations. However, there is considerable related analysis, which supports our narrative, and can be found within a broad range of sources, which we describe next.

The author's approximate DP/RL books

[1] Bertsekas, D. P., 2019. Reinforcement Learning and Optimal Control, Athena Scientific, Belmont, MA.

[2] Bertsekas, D. P., 2020. Rollout, Policy Iteration, and Distributed Reinforcement Learning, Athena Scientific, Belmont, MA.

provide a far more detailed discussion of MPC, adaptive control, discrete optimization, and distributed computation topics, than the present book. Moreover, some other popular methods, such as temporal difference algorithms and Q-learning, are discussed in the books [1] and [2], but not in the present book.

The author's two-volume DP book

[3] Bertsekas, D. P., 2017. Dynamic Programming and Optimal Control, Vol. I, 4th Edition, Athena Scientific, Belmont, MA.

[4] Bertsekas, D. P., 2012. Dynamic Programming and Optimal Control, Vol. II, 4th Edition, Athena Scientific, Belmont, MA.

is a major source on the modeling and mathematical aspects of finite and infinite horizon DP. Modeling aspects and finite horizon problems are the principal focus of [3], while the mathematical aspects of infinite horizon problems are the principal focus of [4]. Both books [3] and [4] provide substantial accounts of approximate DP/RL methods. Thus these books are the best entry points for a research-oriented reader that wishes to go into DP and its connections to RL more deeply.

Two of the author's recent research monographs are also highly relevant to our narrative:

[5] Bertsekas, D. P., 2022. Abstract Dynamic Programming, 3rd Ed., Athena Scientific, Belmont, MA (can be downloaded from the author's website).

This monograph focuses on the analytical aspects of abstract DP on which the Newton-based methodology is couched, and may serve as a mathematical supplement to the present book. It also provides some supportive mathematical foundation for the more visually oriented monograph

[6] Bertsekas, D. P., 2022. Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control, Athena Scientific, Belmont, MA (can be downloaded from the author's website).

This monograph relies on intuitive visualization to convey the main ideas of our off-line training/on-line play/Newton's method conceptual framework for approximation in value space. It also focuses on model predictive and adaptive control, and associated issues of stability. Like our Section 1.5, it is visually oriented, but goes into greater detail into various special cases, including pathological exceptions.

All of the above books are available as ebooks as well as in print form; see the Athena Scientific website. A lot of the material in this book is adapted from these books. However, the books themselves collectively provide a far more detailed and mathematically rigorous presentation.

An extensive overview of the connections and applications of the conceptual framework of this book with model predictive and adaptive control is given in the paper

[7] Bertsekas, D. P., 2024. "Model Predictive Control, and Reinforcement Learning: A Unified Framework Based on Dynamic Programming," published in Proc. IFAC NMPC (can be downloaded from the author's website; a videolecture is available on-line).

Together with Chapter 1 of this book, it can be used as a starting point for a course in modern control system design.

The present book can also be fruitfully supplemented by the extensive textbook and research monograph literature on RL. This literature is summarized in Section 1.8, and includes several accounts of RL that are based on alternative viewpoints of artificial intelligence, control theory, and operations research.

Structure of the Book - Course Adaptations

A key structural feature of this book is its modular organization, with a view towards flexibility, so it can be easily modified to accommodate changes in course content. In particular, the book is divided in two parts:

(1) A *foundational platform*, which consists of Chapter 1. It provides a selective overview of the approximate DP/RL landscape, and a

starting point for a more detailed in-class development of other RL topics, whose choice can be at the instructor's discretion.

(2) An *in-depth coverage*, which consists of Chapters 2 and 3. It offers a deeper focus into specific methodologies. In particular, Chapter 2 describes primarily deterministic and stochastic rollout techniques, as well as some related approximation in value methods. Chapter 3 addresses the use of neural networks and other approximation architectures for off-line training.

This modular structure allows for customization based on a course's focus. For example, instructors can use the foundational platform of Chapter 1 to build either more mathematically-oriented or less formal courses, depending on the needs of their class.

Let us also mention that the book contains more material than can be reasonably covered in class in one semester. This provides some flexibility to an instructor regarding the choice of material to present.

Videolectures and Slides

The present book and my RL books above, were developed while teaching several versions of my course at ASU. Videolectures and slides from this course, as well as links to overview videolectures by the author are available from my website

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

and provide a good supplement and companion resource to this book.

Thanks and Appreciation

The hospitable and stimulating environment at ASU contributed much to shaping my course during the period 2019-2024. I am thankful to several colleagues and students, and to my teaching assistants, Sushmita Bhattacharya, Sahil Badyal, and Jamison Weber, for their comments and suggestions. Very special thanks are due to Yuchao Li for many valuable interactions, collaborative research, and proofreading support.

1

Exact and Approximate Dynamic Programming

Contents

This chapter has multiple purposes:

(a) *To provide an overview of the exact dynamic programming (DP) methodology, with a focus on suboptimal solution methods*. We will first discuss finite horizon problems, which involve a finite sequence of successive decisions, and are thus conceptually and analytically simpler. We will consider separately deterministic and stochastic finite horizon 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 reinforcement learning (RL) methods that are the main subject of the book. We will also discuss somewhat briefly the more intricate infinite horizon methodology (Section 1.4), and refer to the author's DP textbooks [Ber12], [Ber17a], the RL books [Ber19a], [Ber20a], and the neuro-dynamic programming monograph [BeT96] for a fuller presentation.

(b) *To summarize the principal RL methodologies, with primary emphasis on approximation in value space*. This is the approximate DP-based architecture that underlies the AlphaZero, AlphaGo, TD-Gammon and other related programs, as well as the Model Predictive Control (MPC) methodology, one of the principal control system design methods. We will also argue later (Chapter 2) that approximation in value space provides the entry point for the use of RL methods for solving discrete optimization and integer programming problems.

(c) *To explain the major principles of approximation in value space, and its division into the off-line training and the on-line play algorithms*. A key idea here is the connection of these two algorithms through the algorithmic methodology of Newton's method for solving the problem's Bellman equation. This viewpoint, recently developed in the author's "Rollout and Policy Iteration ..." book [Ber20a] and the visually oriented "Lessons from AlphaZero ..." monograph [Ber22a], underlies the entire course and is discussed for the simple, intuitive, and important class of linear quadratic problems in Section 1.5.

(d) *To overview the range of problem types where our RL methods apply, and to explain some of their major algorithmic ideas (Section 1.6)*. Included here are partial state observation problems (POMDP), multiagent problems, and problems with unknown model parameters, which can be addressed with adaptive control methods.

This chapter will also discuss selectively some major algorithms in approximate DP and RL, including rollout and policy iteration. A broader discussion of DP/RL may be found in the RL books [Ber19a], [Ber20a], the DP textbooks [Ber12], [Ber17a], the neuro-dynamic programming mono-

graph [BeT96], alongside other textbooks referenced in the final section.

The book reflects the author's decision/control and operations research orientation, which has in turn guided the choices of terminology, notation, and mathematical style throughout. On the other hand, RL methods have been developed within the artificial intelligence community, as well as the decision/control and operations research communities. Despite similarities in the mathematical structures of the problems that these communities address, there are notable differences in terminology, notation, and culture, which can be confusing to researchers entering the field. We have thus provided in Section 1.7 a glossary and an orientation to assist the reader in navigating the full range of the DP/RL literature.

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 2022, 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 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 theory and 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.

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], who mention several related antecedent works. For a better understanding of the connections of AlphaZero

and AlphaGo Zero with Tesauro's programs and the concepts developed here, the "Methods" section of the paper [SSS17] is recommended.

To understand the overall structure of AlphaZero and its connection to our DP/RL methodology, it is useful to divide its design into two parts: *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, and *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

This is the part of the program that learns how to play through off-line self-training, and is illustrated in Fig. 1.1.1. The algorithm generates a sequence of *chess players* and *position evaluators*. A chess player assigns "probabilities" to all legal moves in a given position, representing the likelihood of each move being chosen. A position evaluator assigns a numerical score to a given position, predicting the player's chances of winning from that position. The chess player and the position evaluator are represented by two neural networks, a *policy network* and a *value network*, which accept a chess position and generate a set of move probabilities and a position evaluation, respectively.†

In more traditional DP terminology, a position is the state of the game, a position evaluator is a cost function that gives (an estimate of) the optimal 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 classical 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"

† Here the neural networks play the role of *function approximators*; see Chapter 3. 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 are also discussed in the RL books [Ber19a], [Ber20a], and the neuro-dynamic programming book [BeT96]).

‡ 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 extends to two-player games, although we will not focus on this extension. Alternately, we can consider training a game program to play against a known fixed opponent; this is a one-player setting, which is discussed further in Section 2.12.

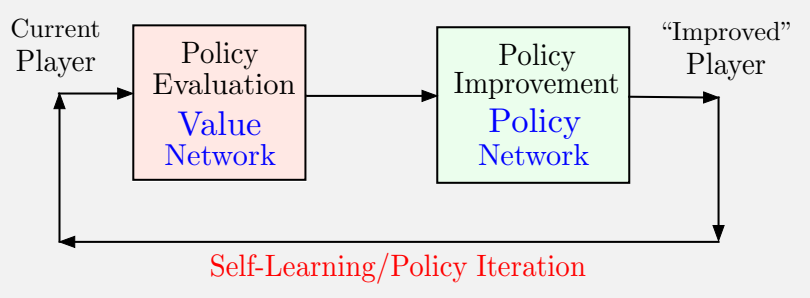

Figure 1.1.1 Illustration of the AlphaZero training algorithm. It generates a sequence of position evaluators and chess players. The position evaluator and the chess player are represented by two neural networks, a value network and a policy network, which accept a chess position and generate a position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

out of all the players generated.† Policy iteration may be separated conceptually in 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 collected and used to train a value network, which then serves as the position evaluator for the player.

(b) *Policy improvement*: Given the current player and its position evaluator, trial move sequences are selected and evaluated for the rest 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*. However, policy improvement can also be done more simply. For example, all possible move sequences from a given position could be tried, extending forward a few moves, with the terminal

† 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." Note, however, that a two-person game player, trained through selfplay, may fail against a particular human or computer player that can exploit training vulnerabilities. This is a theoretical but rare possibility; see our discussion in Section 2.12.

position evaluated by 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.

Tesauro's TD-Gammon algorithm [Tes94] is similarly based on approximate policy iteration, but uses a different methodology for approximate policy evaluation, based on the TD(λ) algorithm. Unlike AlphaZero, TD-Gammon does not employ a policy network or MCTS. Instead, it relies solely on a value network, to replicate the functionality of a policy network by generating moves on-line via a one-step or two-step lookahead minimization. For a detailed description, see Section 8.6 of the neuro-dynamic programming book [BeT96].

On-Line Play and Approximation in Value Space - Rollout

Suppose that a "final" player has been obtained through the AlphaZero off-line training process just described. This player could, in principle, play chess against any human or computer opponent by generating move probabilities using its off-line-trained policy network. At any position, the player would simply select the move with the highest probability from the policy network. While this approach would allow the player to make decisions quickly, it would not be strong enough to defeat highly skilled human opponents. AlphaZero's extraordinary strength arises only when the off-line-trained player and position evaluator are embedded into another algorithm, which we refer to as the on-line player (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 evaluates the effect of the remaining moves by using the position evaluator of the off-line obtained value network.

The architecture of the final version of Tesauro's TD-Gammon program [TeG96] is similar to the one of AlphaZero, and uses an off-line neural network-trained terminal position evaluator; see Fig. 1.1.3. However, it also includes a middle portion, called "truncated rollout," which involves running a player for a few moves, before using the position evaluator. In effect, rollout can be viewed as an economical way to extend the length of the lookahead. In the published version of AlphaZero/chess [SHS17], there is also a rollout portion, but it is is rather rudimentary; the first portion (multistep lookahead) is quite long and efficiently implemented, so that a sophisticated rollout portion is not essential. Rollout plays a significant role in AlphaGo [SHM16], and is critically important in Tesauro's backgammon program, where long multistep lookahead is infeasible due to the rapid expansion of the lookahead tree.

Architectures that are similar to the ones of AlphaZero and TD-Gammon will be generically referred to as *approximation in value space*

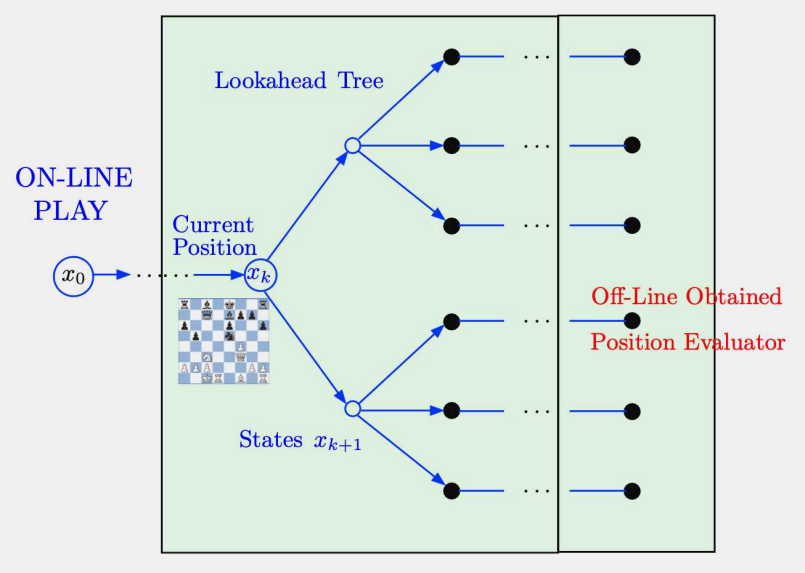

Figure 1.1.2 Illustration of the on-line player of AlphaZero and many other computer chess programs. At a given position, it generates a lookahead tree of multiple moves up to some depth, and evaluates the effect of the remaining moves by using the position evaluator of the off-line player. There are many implementation details that we do not discuss here; for example the lookahead is selective, because some lookahead paths are pruned, by using a form of Monte Carlo tree search. Note that the off-line-trained neural network of AlphaZero produces both a position evaluator and a playing policy. However, the neural network-trained policy is not used directly for on-line play.

in this book. Architectures of this type are also known as *approximate dynamic programming*, or *neuro-dynamic programming*, and will be central for our purposes.†

Among other settings, approximation in value space is used in control system design, particularly in *model predictive control* (MPC), which is briefly described in Section 6.1.9. There, the number of steps in lookahead

† The names "approximate dynamic programming" and "neuro-dynamic programming" are often used as synonyms to RL. However, RL is generally thought to also subsume the methodology of approximation in policy space, which involves search for optimal parameters within a parametrized set of policies. The search is done with methods that are largely unrelated to DP, such as for example stochastic gradient or random search methods. Approximation in policy space may be used off-line to design a policy that can be used for on-line rollout. It will be discussed rather briefly in this book (see Sections 3.4 and 3.5). A more detailed discussion, consistent with the terminology used here, can be found in Chapter 5 of the RL book [Ber19a].

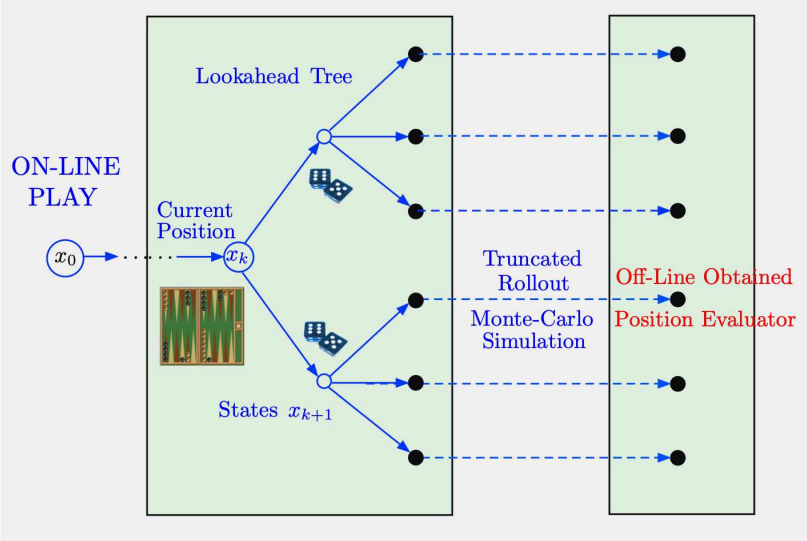

Figure 1.1.3 Illustration of the architecture of TD-Gammon with truncated rollout [TeG96]. It uses a relatively short lookahead minimization followed by rollout and terminal position evaluation (i.e., game simulation with the one-step lookahead player; the game is truncated after a number of moves, with a position evaluation at the end). Note that backgammon involves stochastic uncertainty, and its state is the pair of current board position and dice roll. For this reason, rollout requires Monte-Carlo simulation, and is quite time-consuming. As a result, backgammon programs that play under restrictive time controls, use limited or highly truncated forms of rollout.

minimization is called the *control interval*, while the total number of steps in lookahead minimization and truncated rollout is called the *prediction interval*; see e.g., Magni et al. [MDM01].† The benefit of truncated rollout in providing an economical substitute for longer lookahead minimization is well known in MPC.

It should be noted that the preceding description of AlphaZero and related approximation in value space architectures is somewhat simplified. We will be discussing 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 to the book. Moreover, the algorithmic division between off-line training and on-line policy implementation will be conceptually very important for our discussions.

We finally note that in approximation in value space the off-line training and the on-line play algorithms can often be decoupled and independently designed. For example the off-line training portion may be simple,

† The Matlab toolbox for MPC design explicitly allows the user to set these two intervals.

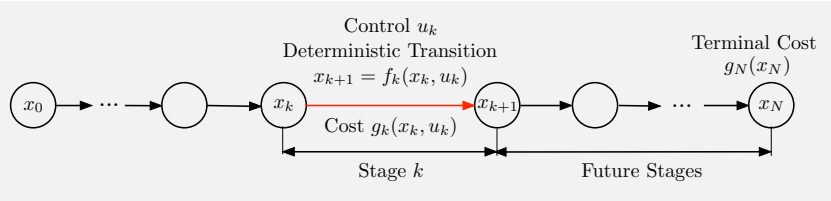

Figure 1.2.1 Illustration of a deterministic N-stage optimal control problem. Starting from state x_k, the next state under control u_k is generated nonrandomly, according to

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

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

such as using a known 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 actions or controls. 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; see Fig. 1.2.1. 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,

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,

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.

In the case of a finite number of states, the system function f_k may be represented by a table that gives the next state x_{k+1} for each possible value of the pair (x_k, u_k). Otherwise a mathematical expression or a computer implementation is necessary to represent f_k.

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 scalar 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

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

where $g_N(x_N)$ is a terminal cost incurred at the end of the process. This is a well-defined scalar, 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). 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:‡

$$J^*(x_0) = \min_{\substack{u_k \in U_k(x_k) \\ k=0,\ldots,N-1}} J(x_0; u_0, \ldots, u_{N-1}). \tag{1.3}$$

† 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 general 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 Section 1.7.2). 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).

‡ 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.

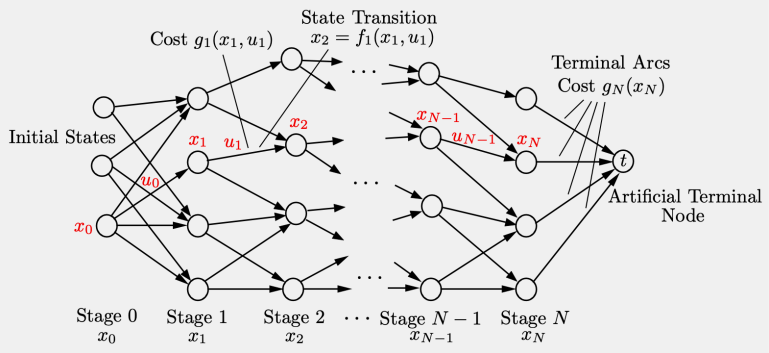

Figure 1.2.2 Transition graph for a deterministic finite-state system. Nodes correspond to states x_k. Arcs correspond to state-control pairs (x_k, u_k). An arc (x_k, u_k) has start and end nodes x_k and $x_{k+1} = f_k(x_k, u_k)$, respectively. We view the cost $g_k(x_k, u_k)$ of the transition as the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

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 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 problems. The idea

† 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.

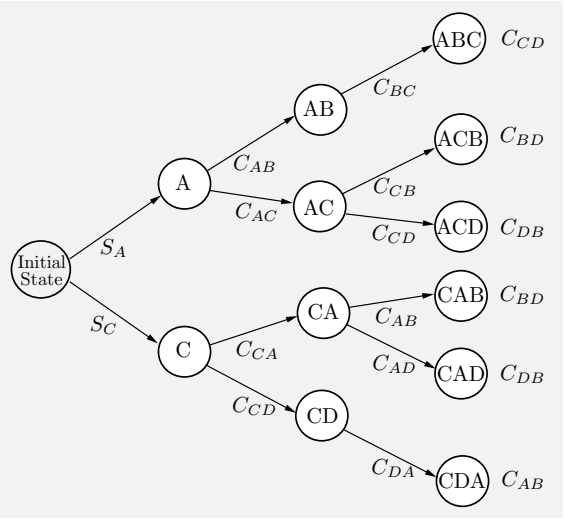

Figure 1.2.3 The transition graph of the deterministic scheduling problem of Example 1.2.1. Each arc of the graph corresponds to a decision leading from some state (the start node of the arc) to some other state (the end node of the arc). The corresponding cost is shown next to the arc. The cost of the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

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 given 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 is not.) The setup cost C_{mn} for passing from any operation m to any other operation n is given. There is also an initial startup cost S_A or S_C for starting with operation A or C, respectively (cf. Fig. 1.2.3). 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 (which in turn is a special case of the graph of Fig. 1.2.2). 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. Generally, DP is used to solve a problem of sequential decision making over N stages, by breaking it down to a sequence of simpler single-stage problems.

In particular, the algorithm aims to find a sequence of optimal controls $u_0^*, \ldots, u_{N-1}^*$ by generating a corresponding sequence of optimal cost functions $J_0^*, \ldots, J_{N-1}^*$. It starts with J_N^* equal to the terminal cost function g_N and computes the next function J_{N-1}^*, by solving a single stage decision problem whose optimization variable is u_{N-1}. It then uses J_{N-1}^* to compute J_{N-2}^*, and proceeds similarly to compute all the remaining cost functions $J_{N-3}^*, \ldots, J_0^*$.

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^* (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

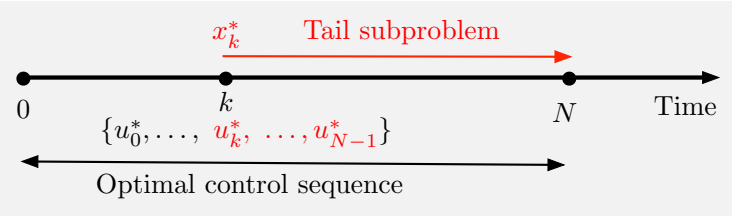

Figure 1.2.4 Schematic illustration of the principle of optimality. The tail $\{u_k^*, \ldots, u_{N-1}^*\}$ of an optimal sequence $\{u_0^*, \ldots, u_{N-1}^*\}$ is optimal for the tail subproblem that starts at the state x_k^* of the optimal state trajectory.

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 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

† 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."

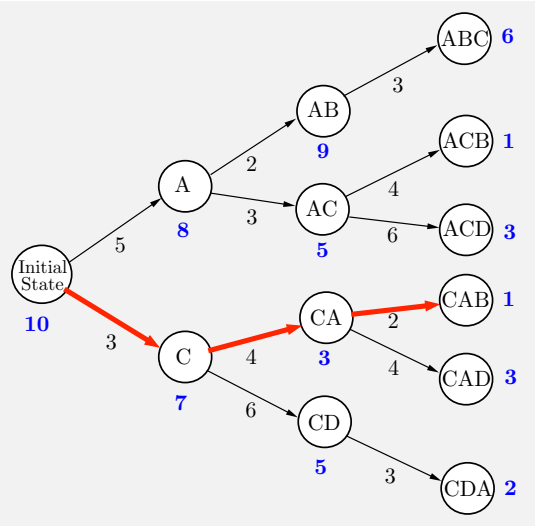

Figure 1.2.5 Transition graph of the deterministic scheduling problem, with the cost of each decision shown next to the corresponding arc. Next to each node/state we show the cost to optimally complete the schedule starting from that state. This is the optimal cost of the corresponding tail subproblem (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.

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. We will show that the value $J_k^*(x_k)$ represents the optimal cost of the tail subproblem that starts at state x_k at time k.

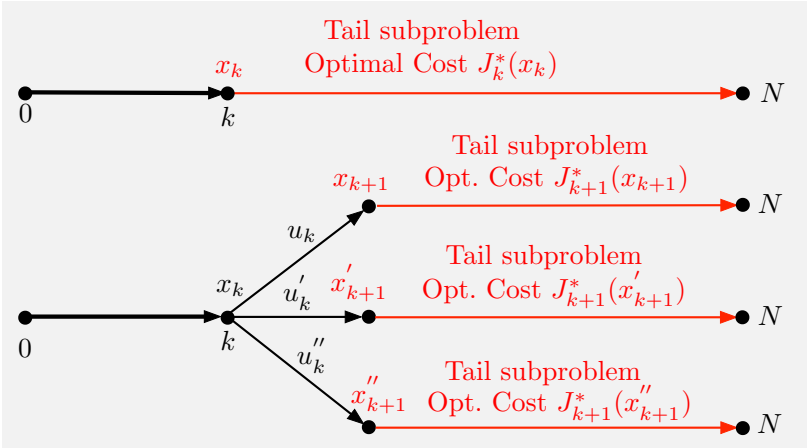

Figure 1.2.6 Illustration of the DP algorithm. The tail subproblem that starts at x_k at time k minimizes over $\{u_k, \ldots, u_{N-1}\}$ the "cost-to-go" from k to N,

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

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

$$J_N^*(x_N) = g_N(x_N), \qquad \text{for all } x_N, \tag{1.4}$$

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

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big) \Big], \qquad \text{for all } x_k. \tag{1.5}$$

The DP algorithm together with the construction of the functions $J_k^*(x_k)$ are illustrated in Fig. 1.2.6. Note that at stage k, the calculation in Eq. (1.5) 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

$$J_k^*(x_k) = \min_{\substack{u_m \in U_m(x_m) \\ m=k,\ldots,N-1}} J(x_k; u_k, \ldots, u_{N-1}), \tag{1.6}$$

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

$$J(x_k; u_k, \ldots, u_{N-1}) = g_N(x_N) + \sum_{t=k}^{N-1} g_t(x_t, u_t). \tag{1.7}$$

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 (1.6) 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 (1.5) 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 corresponding state trajectory $\{x_1^*, \ldots, x_N^*\}$ for the given initial state x_0.

---

† 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. (1.6) and (1.7) to write

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

where for the last equality we use the induction hypothesis. A subtle mathematical point here is that, through the minimization operation, the cost-to-go functions J_k^* may take the value $-\infty$ for some x_k. Still the preceding induction argument is valid even if this is so.

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

$$u_k^* \in \arg \min_{u_k \in U_k(x_k^*)} \Big[g_k(x_k^*, u_k) + J_{k+1}^*\big(f_k(x_k^*, u_k)\big) \Big], \qquad (1.8)$$

and

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

The same algorithm can be used to find an optimal control sequence for any tail subproblem. 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.

The following example deals with the classical traveling salesman problem involving N cities. Here, the number of states grows exponentially with N, and so does the corresponding amount of computation for exact DP. We will show later that with rollout, we can solve the problem approximately with computation that grows polynomially with N.

Example 1.2.2 (The 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.

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 travel-

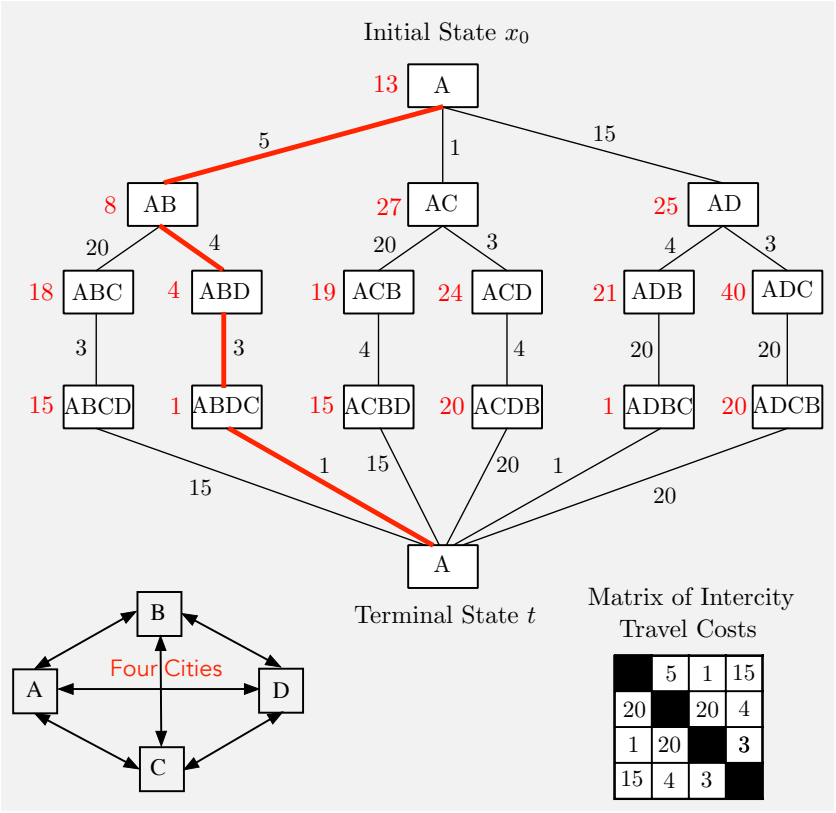

Figure 1.2.7 The DP formulation of the traveling salesman problem of Example 1.2.2. The travel times between the four cities A, B, C, and D are shown in the matrix at the bottom. We form a graph whose nodes are the k-city sequences and correspond to the states of the kth stage, assuming that A is the 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. (1.8)], starting from the initial state x_0.

ing salesman and related scheduling problems are typically not addressed with exact DP, but rather with approximation methods. Some of these methods are based on DP and will be discussed later.

Q-Factors and Q-Learning

An alternative (and equivalent) form of the DP algorithm (1.5), 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

$$Q_k^*(x_k, u_k) = g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big). \tag{1.9}$$

Thus the optimal Q-factors are simply the expressions that are minimized in the right-hand side of the DP equation (1.5).†

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

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} Q_k^*(x_k, u_k). \tag{1.10}$$

Moreover, the DP algorithm (1.5) can be written in an essentially equivalent form that involves Q-factors only [cf. Eqs. (1.9)-(1.10)]:

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

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*.

1.2.3 Approximation in Value Space and Rollout

The forward optimal control sequence construction of Eq. (1.8) 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 algorithm (1.8).

† The term "Q-factor" has been used in the books [BeT96], [Ber19a], [Ber20a] 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 an influential Ph.D. thesis [Wat89] that proposed the use of Q-factors in RL.

‡ Approximation in value space (sometimes called "search" or "tree search" in the AI literature) 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 conceptually underlies the widely used A^* method for computing approximate solutions to large scale shortest path problems. For a view of A^* that is consistent with our approximate DP framework, the reader may consult the author's DP book [Ber17a].

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

$$\tilde{u}_k \in \arg\min_{u_k \in U_k(\tilde{x}_k)} \Big[g_k(\tilde{x}_k, u_k) + \tilde{J}_{k+1}\big(f_k(\tilde{x}_k, u_k)\big) \Big], \tag{1.11}$$

and

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

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 $\{u_0^*, \ldots, u_{N-1}^*\}$, and is independent of how the functions $\tilde{J}_k$ are computed. The motivation for approximation in value space for stochastic DP problems is vastly reduced computation relative to the exact DP algorithm (once $\tilde{J}_k$ have been obtained): the minimization (1.11) needs to be performed only for the N states $x_0, \tilde{x}_1, \ldots, \tilde{x}_{N-1}$ that are encountered during the on-line control of the system, and not for every state within the potentially enormous state space, as is the case for exact DP.

The algorithm (1.11) is said to involve a *one-step lookahead minimization*, since it solves a one-stage DP problem for each k. In what follows we 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. 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. (1.11), becomes more time consuming.

Rollout with a Base Heuristic for Deterministic Problems

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. Alternatively, $\tilde{J}_k$ may be obtained on-line with *rollout*, which will be discussed in detail in this book. 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 stages, starting from the state x_k, and accumulating the costs incurred at these stages.

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, multistep lookahead, and stochastic uncertainty. We will discuss such variants later. For the moment we will focus on a deterministic DP problem with a finite number of controls. Given a state x_k at time k, this algorithm considers all the tail subproblems that start at every possible next state x_{k+1}, and solves them suboptimally by using some algorithm, referred to as base heuristic.

Thus when at x_k, rollout generates on-line the next states x_{k+1} that correspond to all $u_k \in U_k(x_k)$, and uses the base heuristic to compute the sequence of states $\{x_{k+1}, \ldots, x_N\}$ and controls $\{u_{k+1}, \ldots, u_{N-1}\}$ such that

$$x_{t+1} = f_t(x_t, u_t), \qquad t = k, \ldots, N-1,$$

and the corresponding cost

$$H_{k+1}(x_{k+1}) = g_{k+1}(x_{k+1}, u_{k+1}) + \cdots + g_{N-1}(x_{N-1}, u_{N-1}) + g_N(x_N).$$

The rollout algorithm then applies the control that minimizes over $u_k \in U_k(x_k)$ the tail cost expression for stages k to N:

$$g_k(x_k, u_k) + H_{k+1}(x_{k+1}).$$

† For an intuitive justification of the cost improvement mechanism, note that the rollout control $\tilde{u}_k$ is calculated from Eq. (1.11) 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 explains why multistep lookahead tends to provide better performance than one-step lookahead in rollout schemes.

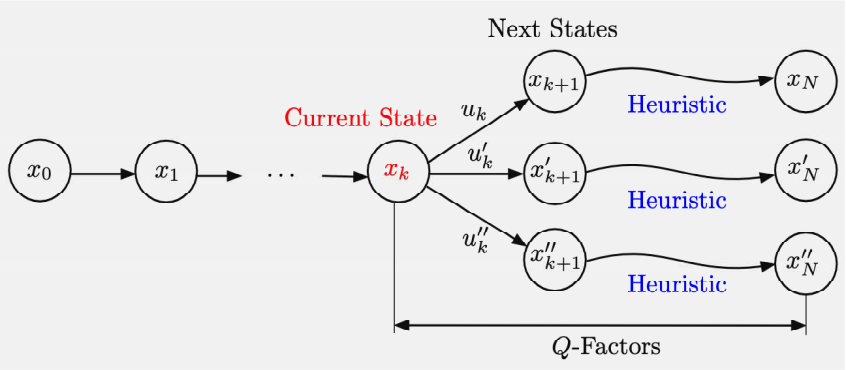

Figure 1.2.9 Schematic illustration of rollout with one-step lookahead for a deterministic problem. At state x_k, for every pair (x_k, u_k), $u_k \in U_k(x_k)$, the base heuristic generates an approximate Q-factor

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

and selects the control $\tilde{\mu}_k(x_k)$ with minimal Q-factor.

Equivalently, and more succinctly, the rollout algorithm applies at state x_k the control $\tilde{\mu}_k(x_k)$ given by the minimization

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k), \tag{1.12}$$

where $\tilde{Q}_k(x_k, u_k)$ is the approximate Q-factor defined by

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big); \tag{1.13}$$

see Fig. 1.2.9.

Note that the rollout algorithm requires running the base heuristic for a number of times that is bounded by Nn, where n is an upper bound on the number of control choices available at each state. Thus if n is small relative to N, it requires computation equal to a small multiple of N times the computation time for a single application of the base heuristic. Similarly, if n is bounded by a polynomial in N, the ratio of the rollout algorithm computation time to the base heuristic computation time is a polynomial in N.

Example 1.2.3 (Traveling Salesman Problem)

Let us consider the traveling salesman problem of Example 1.2.2, whereby a salesman wants to find a minimum cost tour that visits each of N given cities $c = 0, \ldots, N-1$ exactly once and returns to the city he started from. With each pair of distinct cities c, c', we associate a traversal cost $g(c, c')$. Note

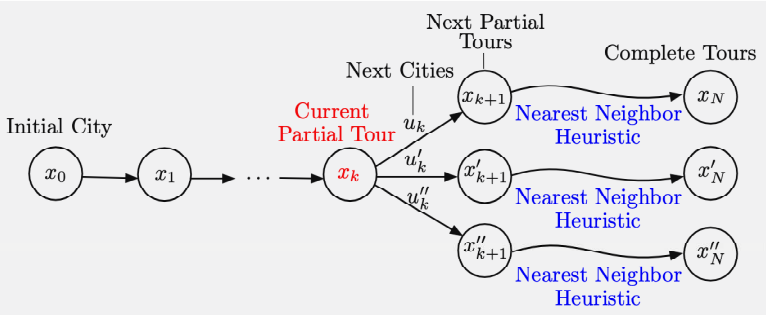

Figure 1.2.10 Rollout with the nearest neighbor heuristic for the traveling salesman problem of Example 1.2.3. The initial state x_0 consists of a single city. The final state x_N is a complete tour of N cities, containing each city exactly once.

that we assume that we can go directly from every city to every other city. There is no loss of generality in doing so because we can assign a very high cost $g(c, c')$ to any pair of cities (c, c') that is precluded from participation in the solution. The problem is to find a visit order that goes through each city exactly once and whose sum of costs is minimum.

There are many heuristic approaches for solving the traveling salesman problem. For illustration purposes, let us focus on the simple *nearest neighbor* heuristic, which starts with a partial tour, i.e., an ordered collection of distinct cities, and constructs a sequence of partial tours, adding to the each partial tour a new city that does not close a cycle and minimizes the cost of the enlargement. In particular, given a sequence $\{c_0, c_1, \ldots, c_k\}$ (with $k < N-1$) consisting of distinct cities, the nearest neighbor heuristic adds a city c_{k+1} that minimizes $g(c_k, c_{k+1})$ over all cities $c_{k+1} \neq c_0, \ldots, c_k$, thereby forming the sequence $\{c_0, c_1, \ldots, c_k, c_{k+1}\}$. Continuing in this manner, the heuristic eventually forms a sequence of N cities, $\{c_0, c_1, \ldots, c_{N-1}\}$, thus yielding a complete tour with cost

$$g(c_0, c_1) + \cdots + g(c_{N-2}, c_{N-1}) + g(c_{N-1}, c_0). \tag{1.14}$$

We can formulate the traveling salesman problem as a DP problem as we discussed in Example 1.2.2. We choose a starting city, say c_0, as the initial state x_0. Each state x_k corresponds to a partial tour $(c_0, c_1, \ldots, c_k)$ consisting of distinct cities. The states x_{k+1}, next to x_k, are sequences of the form $(c_0, c_1, \ldots, c_k, c_{k+1})$ that correspond to adding one more unvisited city $c_{k+1} \neq c_0, c_1, \ldots, c_k$ (thus the unvisited cities are the feasible controls at a given partial tour/state). The terminal states x_N are the complete tours of the form $(c_0, c_1, \ldots, c_{N-1}, c_0)$, and the cost of the corresponding sequence of city choices is the cost of the corresponding complete tour given by Eq. (1.14). Note that the number of states at stage k increases exponentially with k, and so does the computation required to solve the problem by exact DP.

Let us now use as a base heuristic the nearest neighbor method. The corresponding rollout algorithm operates as follows: After $k < N-1$ iterations, we have a state x_k, i.e., a sequence $\{c_0, \ldots, c_k\}$ consisting of distinct cities. At the next iteration, we add one more city by running the

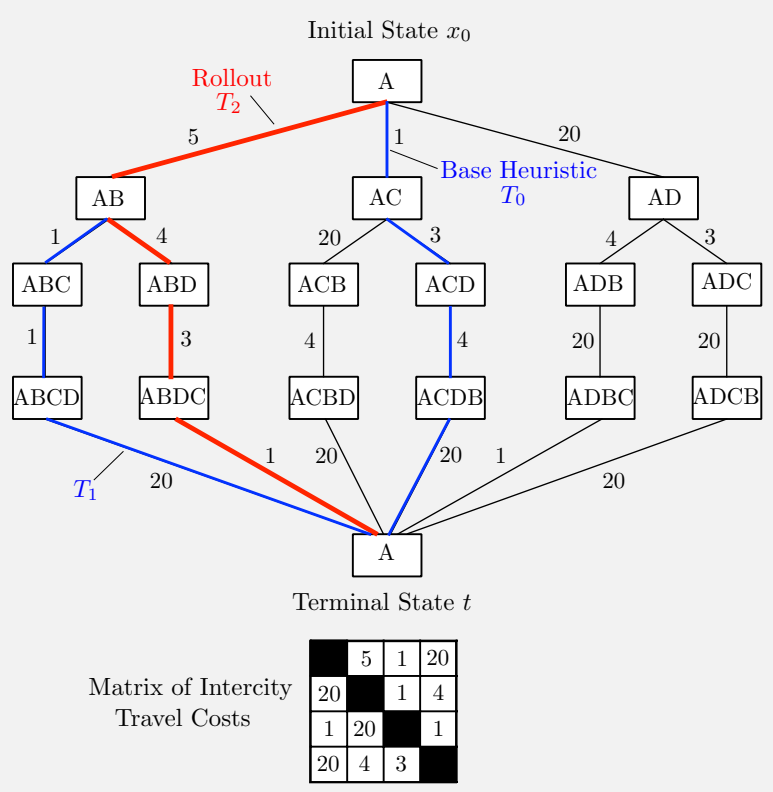

Figure 1.2.11 Rollout with the nearest neighbor base heuristic, applied to a traveling salesman problem. At city A, the nearest neighbor heuristic generates the tour ACDBA (labelled T_0). At city A, the rollout algorithm compares the tours ABCDA, ACDBA, and ADCBA, finds ABCDA (labelled T_1) to have the least cost, and moves to city B. At AB, the rollout algorithm compares the tours ABCDA and ABDCA, finds ABDCA (labelled T_2) to have the least cost, and moves to city D. The rollout algorithm then moves to cities C and A (it has no other choice). The final tour T_2 generated by rollout turns out to be optimal in this example, while the tour T_0 generated by the base heuristic is suboptimal. This is suggestive of a general result: the rollout algorithm for deterministic problems generates a sequence of solutions of decreasing cost under some conditions on the base heuristic that we will discuss in Chapter 2, and which are satisfied by the nearest neighbor heuristic.

nearest neighbor heuristic starting from each of the sequences of the form $\{c_0, \ldots, c_k, c\}$ where $c \neq c_0, \ldots, c_k$. We then select as next city c_{k+1} the city c that yielded the minimum cost tour under the nearest neighbor heuristic; see Fig. 1.2.10. The overall computation for the rollout solution is bounded by a polynomial in N, and is much smaller than the exact DP computation. Figure 1.2.11 provides an example where the nearest neighbor heuristic and the corresponding rollout algorithm are compared; see also Exercise 1.1.

1.3 STOCHASTIC EXACT AND APPROXIMATE 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.

1.3.1 Finite Horizon Problems

The stochastic optimal control problem differs from its deterministic counterpart 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, 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

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

† The discrete equation format and corresponding x-u-w notation is standard in the optimal control literature. For finite-state stochastic problems, also called *Markovian Decision Problems* (MDP), the system is often represented conveniently in terms of control-dependent transition probabilities. A common notation in the RL literature is $p(s, a, s')$ for transition probability from s to s' under action a. This type of notation is not well suited for deterministic problems, which involve no probabilistic structure at all and are of major interest in this book. The transition probability notation is also cumbersome for problems with a continuous state space; see Sections 1.7.1 and 1.7.2 for further discussion. The reader should note, however, that mathematically the system equation and transition probabilities are equivalent, and any analysis that can be done in one notational system can be translated to the other notational system.

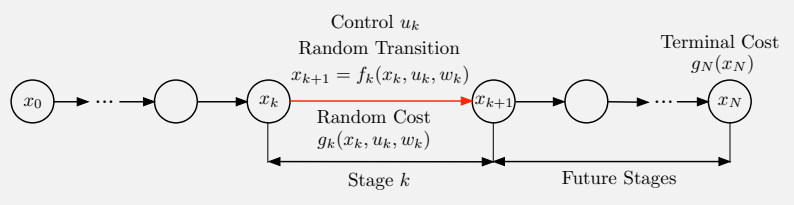

Figure 1.3.1 Illustration of an N-stage stochastic optimal control problem. Starting from state x_k, the next state under control u_k is generated randomly, according to $x_{k+1} = f_k(x_k, u_k, w_k)$, where w_k is the random disturbance, and a random stage cost $g_k(x_k, u_k, w_k)$ is incurred.

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^* 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 $\{u_0, \ldots, u_{N-1}\}$ [cf. Eq. (1.3)], 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.

† 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 textbook by Bertsekas and Tsitsiklis [BeT08].

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*.

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

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\Big\}. \tag{1.15}$$

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. (1.15). 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 (1.15) on-line.

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 using the stochastic DP algorithm. 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)

Here the system is linear,

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

and the state, control, and disturbance are scalars. The cost is quadratic of the form:

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

where q and r are known positive weighting parameters. 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).

As an illustration, 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

$$v_{k+1} = v_k + bu_k + w_k, \tag{1.16}$$

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, the road conditions. The cost function expresses our desire to keep x_k near zero with relatively little force.

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

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

and by applying Eq. (1.15), we obtain

$$\begin{aligned} J^*_{N-1}(x_{N-1}) &= \min_{u_{N-1}} E\Big\{qx^2_{N-1} + ru^2_{N-1} + J^*_N(ax_{N-1} + bu_{N-1} + w_{N-1})\Big\} \\ &= \min_{u_{N-1}} E\Big\{qx^2_{N-1} + ru^2_{N-1} + q(ax_{N-1} + bu_{N-1} + w_{N-1})^2\Big\} \\ &= \min_{u_{N-1}} \Big[qx^2_{N-1} + ru^2_{N-1} + q(ax_{N-1} + bu_{N-1})^2 \\ &\qquad + 2qE\{w_{N-1}\}(ax_{N-1} + bu_{N-1}) + qE\{w^2_{N-1}\}\Big], \end{aligned}$$

and finally, using the assumptions $E\{w_{N-1}\} = 0$, $E\{w^2_{N-1}\} = \sigma^2$, and bringing out of the minimization the terms that do not depend on u_{N-1},

$$J^*_{N-1}(x_{N-1}) = qx^2_{N-1} + q\sigma^2 + \min_{u_{N-1}} \Big[ru^2_{N-1} + q(ax_{N-1} + bu_{N-1})^2\Big]. \quad (1.17)$$

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},

$$0 = 2ru_{N-1} + 2qb(ax_{N-1} + bu_{N-1}),$$

we obtain the optimal policy for the last stage:

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

Substituting this expression into Eq. (1.17), we obtain with a straightforward calculation

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

where

$$K_{N-1} = \frac{a^2rq}{r + b^2q} + q.$$

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 (plus constant) 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 obtain the optimal policy and optimal cost-to-go function in the form

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

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

where

$$L_k = -\frac{abK_{k+1}}{r + b^2K_{k+1}}, \qquad k = 0, 1, \ldots, N-1, \quad (1.18)$$

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

$$K_k = \frac{a^2 r K_{k+1}}{r + b^2 K_{k+1}} + q, \qquad k = 0, 1, \ldots, N-1, \tag{1.19}$$

starting from the terminal condition $K_N = q$.

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. This is an important part of the RL methodology, which we will discuss later in this chapter, and in more detail in Chapter 2.

Note that the linear quadratic type of problem illustrated in the preceding example is exceptional in that it admits an elegant analytical solution. Most DP problems encountered in practice require a computational solution.

Q-Factors and Q-Learning for Stochastic Problems

Similar to the case of deterministic problems [cf. Eq. (1.9)], we can define optimal Q-factors for a stochastic problem, as the expressions that are minimized in the right-hand side of the stochastic DP equation (1.15). They are given by

$$Q_k^*(x_k, u_k) = E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\Big\}. \tag{1.20}$$

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

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

We will later be interested in approximate Q-factors, where J_{k+1}^* in Eq. (1.20) is replaced by an approximation $\tilde{J}_{k+1}$. Again, the Q-factor corresponding to a state-control pair (x_k,u_k) is the sum of the expected first stage cost using (x_k,u_k), plus the expected cost of the remaining stages starting from the next state as estimated by the function $\tilde{J}_{k+1}$.

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. (1.8) and (1.11).

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

At any state x_k encountered at stage k, set

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k\in U_k(x_k)} E_{w_k}\Big\{g_k(x_k,u_k,w_k) + \tilde{J}_{k+1}\big(f_k(x_k,u_k,w_k)\big)\Big\}. \tag{1.21}$$

The one-step lookahead minimization (1.21) 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. By contrast, exact DP requires that this type of minimization be done for every state and stage.

The Three Approximations

When designing approximation in value space schemes, one may consider several interesting simplification ideas, which are aimed at alleviating the computational overhead. Aside from cost function approximation (use $\tilde{J}_{k+1}$ in place of J_{k+1}^*), there are other possibilities. One of them is to simplify the lookahead minimization over $u_k \in U_k(x_k)$ [cf. Eq. (1.15)] by replacing $U_k(x_k)$ with a suitably chosen subset of controls that are viewed as most promising based on some heuristic criterion.

In Section 1.6.5, we will discuss a related idea for control space simplification for the multiagent case where the control consists of multiple

components, $u_k = (u_k^1, \ldots, u_k^m)$. Then, a sequence of m single component minimizations can be used instead, with potentially enormous computational savings resulting.

Another type of simplification relates to approximations in the computation of the expected value in Eq. (1.21) by using limited Monte Carlo simulation. The Monte Carlo Tree Search method, which will be discussed in Chapter 2, Section 2.7.4, is one possibility of this type.

Still another type of expected value simplification is based on the *certainty equivalence approach*, which will be discussed in more detail in Chapter 2, Section 2.7.2. In this approach, at stage k, we replace the future random variables $w_{k+1}, \ldots, w_{k+m}$ by some deterministic values $\overline{w}_{k+1}, \ldots, \overline{w}_{k+m}$, such as their expected values. We may also view this as a form of problem approximation, whereby for the purpose of computing $\tilde{J}_{k+1}(x_{k+1})$, we "pretend" that the problem is deterministic, with the future random quantities replaced by deterministic typical values. This is one of the most effective techniques to make approximation in value space for stochastic problems computationally tractable, particularly when it is also combined with multistep lookahead minimization, as we will discuss later.

Figure 1.3.2 illustrates the three approximations involved in approximation in value space for stochastic problems: *cost-to-go approximation, simplified minimization,* and *expected value approximation*. They may be designed largely independently of each other, and may be implemented with a variety of methods. Much of the discussion in this book will revolve around different ways to organize these three approximations for both cases of one-step and multistep lookahead.

As indicated in Fig. 1.3.2, an important approach for cost-to-go approximation is *problem approximation*, whereby the functions $\tilde{J}_{k+1}$ in Eq. (1.21) are obtained as the optimal or nearly optimal cost functions of a simplified optimization problem, which is more convenient for computation. Simplifications may include exploiting decomposable structure, ignoring various types of uncertainties, and reducing the size of the state space. Several types of problem approximation approaches are discussed in the author's RL book [Ber19a]. A major approach is *aggregation*, which will be discussed in Section 3.5. In this book, problem approximation will not receive much attention, despite the fact that it can often be combined very effectively with the approximation in value space methodology that is our main focus.

Another important approach for on-line cost-to-go approximation is rollout, which we discuss next. This is similar to the rollout approach for deterministic problems, discussed in Section 1.2.

Rollout for Stochastic Problems - Truncated Rollout

In the rollout approach, we select $\tilde{J}_{k+1}$ in Eq. (1.21) to be the cost function of a suitable base policy (perhaps with some approximation). Note that

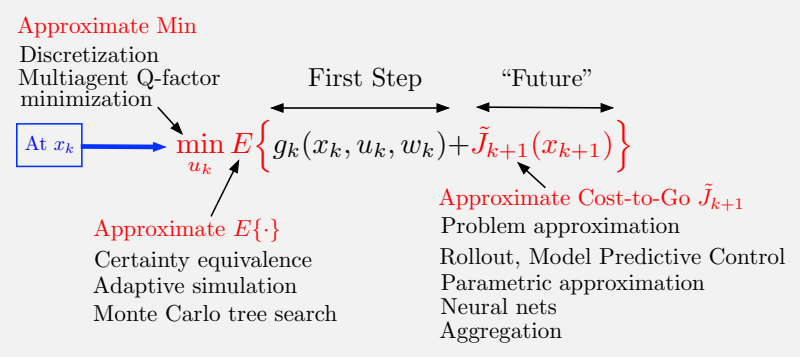

Figure 1.3.2 Schematic illustration of approximation in value space for stochastic problems, and the three approximations involved in its design. Typically the approximations can be designed independently of each other, and with a variety of approaches. There are also multistep lookahead versions of approximation in value space, which will be discussed later.

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 rollout algorithm has a cost improvement property, whereby it yields an improved cost relative to its underlying base policy. We will discuss this property and some conditions under which it is guaranteed to hold in Chapter 2.

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. 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 (1.21); see Fig. 1.3.3. Thus rollout provides an extra layer of lookahead to the one-step minimization, but this lookahead need not extend to the end of the horizon.

Note also that versions of truncated rollout with multistep lookahead minimization are possible. They will be discussed later. 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. Generally, a large combined number of multistep lookahead minimization and rollout steps is likely to be beneficial.

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

Similar to the deterministic case, Q-learning involves the calculation of either the optimal Q-factors (1.20) or approximations $\tilde{Q}_k(x_k, u_k)$. The

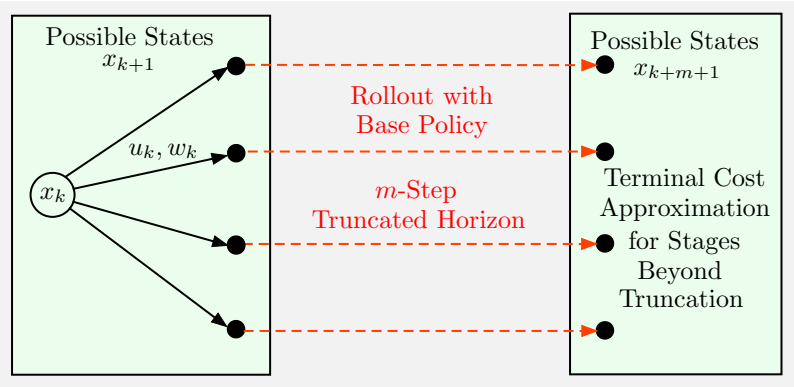

Figure 1.3.3 Schematic illustration of truncated rollout. One-step lookahead is followed by simulation of the base policy for m steps, and an approximate cost $\tilde{J}_{k+m+1}(x_{k+m+1})$ is added to the cost of the simulation, which depends on the state x_{k+m+1} obtained at the end of the rollout. If the base policy simulation is omitted (i.e., $m = 0$), one recovers the general approximation in value space scheme (1.21). Truncated rollout with multistep lookahead is also possible and is discussed in some detail in Chapter 2.

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

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k). \tag{1.22}$$

Since it is possible to implement approximation in value space by using cost function approximations [cf. Eq. (1.21)] or by using Q-factor approximations [cf. Eq. (1.22)], 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

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} E_{w_k} \Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big\}, \tag{1.23}$$

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*. 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 of the approximation in*

value space scheme. By comparison, the Q-factor function approximation scheme (1.22) does not allow for on-line replanning. On the other hand, for problems where there is no need for on-line replanning, the Q-factor approximation scheme may not require the on-line computation of expected values and may allow a much faster on-line computation of the minimizing control $\tilde{\mu}_k(x_k)$ via Eq. (1.22).

One more disadvantage of using Q-factors will emerge later, as we discuss the synergy between off-line training and on-line play based on Newton's method; see Section 1.5. In particular, we will interpret the cost function of the lookahead minimization policy $\{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ as the result of one step of Newton's method for solving the Bellman equation that underlies the DP problem, starting from the terminal cost function approximations $\{\tilde{J}_1, \ldots, \tilde{J}_N\}$. This synergy tends to be negatively affected when Q-factor (rather than cost) approximations are used.

1.3.3 Approximation in Policy Space

The major alternative to approximation in value space is *approximation in policy space*, whereby we select the policy from a suitably restricted class of policies, usually a parametric class of some form. In particular, we can introduce a parametric family of policies (or approximation architecture, as we will call it in Chapter 3),

$$\tilde{\mu}_k(x_k, r_k), \qquad k = 0, \ldots, N-1,$$

where r_k is a parameter, and then estimate the parameters r_k using some type of training process or optimization; cf. Fig. 1.3.4.

Neural networks, described in Chapter 3, are often used to generate the parametric class of policies, in which case r_k is the vector of weights/parameters of the neural network. In Chapter 3, we will also discuss methods for obtaining the training data required for obtaining the parameters r_k, and we will consider several other classes of approximation architectures.

A general scheme for parametric approximation in policy space is to somehow obtain a training set, consisting of a large number of sample state-control pairs (x_k^s, u_k^s), $s = 1, \ldots, q$, such that for each s, u_k^s is a "good" control at state x_k^s. We can then choose the parameter r_k by solving the least squares/regression problem

$$\min_{r_k} \sum_{s=1}^{q} \left\| u_k^s - \tilde{\mu}_k(x_k^s, r_k) \right\|^2 \tag{1.24}$$

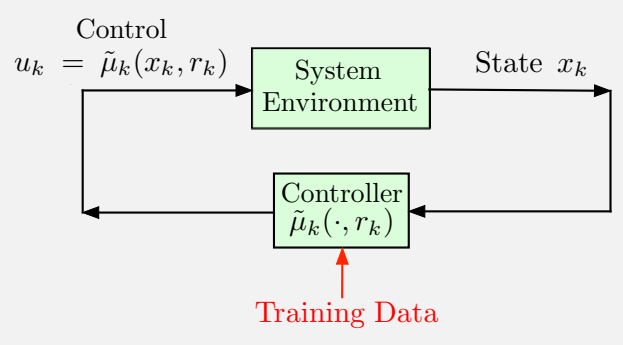

Figure 1.3.4 Schematic illustration of parametric approximation in policy space. A policy

$$\tilde{\mu}_k(x_k, r_k), \qquad k = 0, 1, \ldots, N-1,$$

from a parametric class is computed off-line based on data, and it is used to generate the control $u_k = \tilde{\mu}_k(x_k, r_k)$ on-line, when at state x_k.

(possibly modified to add regularization).† In particular, we may determine u_k^s using a human or a software "expert" that can choose "near-optimal" controls at given states, so $\tilde{\mu}_k$ is trained to match the behavior of the expert. Methods of this type are commonly referred to as *supervised learning* in artificial intelligence.

An important approach for generating the training set (x_k^s, u_k^s), $s = 1, \ldots, q$, for the least squares training problem (1.24) is based on approximation in value space. In particular, we may use a one-step lookahead minimization of the form

$$u_k^s \in \arg\min_{u \in U_k(x_k^s)} E\Big\{ g_k(x_k^s, u, w_k) + \tilde{J}_{k+1}\big(f_k(x_k^s, u, w_k)\big) \Big\},$$

† Here $\|\cdot\|$ denotes the standard quadratic Euclidean norm. It is implicitly assumed here (and in similar situations later) that the controls are members of a Euclidean space (i.e., the space of finite dimensional vectors with real-valued components) so that the distance between two controls can be measured by their normed difference (randomized controls, i.e., probabilities that a particular action will be used, fall in this category). Regression problems of this type arise in the training of *parametric classifiers* based on data, including the use of neural networks (see Section 3.4). Assuming a finite control space, the classifier is trained using the data $\big(x_k^s, u_k^s\big)$, $s = 1, \ldots, q$, which are viewed as state-category pairs, and then a state x_k is classified as being of "category" $\tilde{\mu}_k(x_k, r_k)$. Parametric approximation architectures, and their training through the use of classification and regression techniques are described in Chapter 3. An important modification is to use *regularized regression* where a quadratic regularization term is added to the least squares objective. This term is a positive multiple of the squared deviation $\|r - \hat{r}\|^2$ of r from some initial guess $\hat{r}$.

where $\tilde{J}_{k+1}$ is a suitable (separately obtained) approximation in value space. Alternatively, we may use an approximate Q-factor based minimization

$$u_k^s \in \arg \min_{u_k \in U_k(x_k^s)} \tilde{Q}_k(x_k^s, u_k),$$

where $\tilde{Q}_k$ is a (separately obtained) Q-factor approximation. We may view this as *approximation in policy space built on top of approximation in value space*.

There is a significant advantage of the least squares training procedure of Eq. (1.24), and more generally approximation in policy space: once the parametrized policy $\tilde{\mu}_k$ is obtained, the computation of controls

$$u_k = \tilde{\mu}_k(x_k, r_k), \qquad k = 0, \ldots, N-1,$$

during on-line operation of the system is often much easier compared with the lookahead minimization (1.23). For this reason, one of the major uses of approximation in policy space is to provide an *approximate implementation of a known policy* (no matter how obtained) for the purpose of convenient on-line use. On the negative side, such an implementation is less well suited for on-line replanning.

Model-Free Approximation in Policy Space

There are also alternative optimization-based approaches for policy space approximation. The main idea is that once we use a vector $(r_0, r_1, \ldots, r_{N-1})$ to parametrize the policies π, the expected cost $J_\pi(x_0)$ is parametrized as well, and can be viewed as a function of $(r_0, r_1, \ldots, r_{N-1})$. We can then optimize this cost by using a gradient-like or random search method. This is a widely used approach for optimization in policy space, which, however, will receive limited attention in this book (see Section 3.5, and the RL book [Ber19a], Section 5.7).

An interesting feature of this approach is that in principle it does not require a mathematical model of the system and the cost function; a computer simulator (or availability of the real system for experimentation) suffices instead. This is sometimes called a *model-free implementation*. The advisability of implementations of this type, particularly when they rely exclusively on simulation (i.e., without the use of prior mathematical model knowledge), is a hotly debated and much contested issue; see for example the review paper by Alamir [Ala22].

We finally note an important conceptual difference between approximation in value space and approximation in policy space. The former is primarily an on-line method (with off-line training used optionally to construct cost function approximations for one-step or multistep lookahead). The latter is primarily an off-line training method (which may be used without modification for on-line play or optionally to provide a policy for on-line rollout).

1.3.4 Off-Line Training of Cost Function and Policy Approximations

When it comes to off-line constructed approximations, a major approach is based on the use of parametric approximation. Feature-based architectures and neural networks are very useful within our RL context, and will be discussed in Chapter 3, together with methods that can be used for training them.†

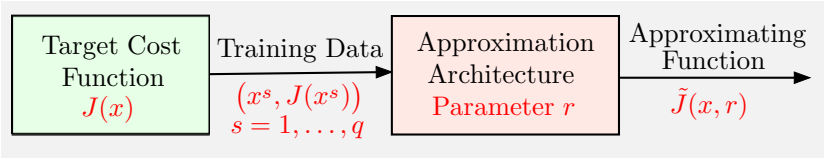

Figure 1.3.5 The general structure for parametric cost approximation. We approximate the target cost function $J(x)$ with a member from a parametric class $\tilde{J}(x, r)$ that depend on a parameter vector r. We use training data $\big(x^s, J(x^s)\big)$, $s = 1, \ldots, q$, and a form of optimization that aims to find a parameter $\hat{r}$ that "minimizes" the size of the errors $J(x^s) - \tilde{J}(x^s, \hat{r})$, $s = 1, \ldots, q$.

A general structure for parametric cost function approximation is illustrated in Fig. 1.3.5. We have a target function $J(x)$ that we want to approximate with a member of a parametric class of functions $\tilde{J}(x, r)$ that depend on a parameter vector r (to simplify, we drop the time index, using J in place of J_k). To this end, we collect training data $\big(x^s, J(x^s)\big)$, $s = 1, \ldots, q$, which we use to determine a parameter $\hat{r}$ that leads to a good "fit" between the data $J(x^s)$ and the predictions $\tilde{J}(x^s, \hat{r})$ of the parametrized function. This is usually done through some form of optimization that

† 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 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 reviewed in Chapter 3. Further DP-oriented discussions are found in many sources, including the RL books [Ber19a], [Ber20a], and the neuro-dynamic programming book [BeT96]. Machine learning books, including those describing at length neural network architectures and training are also recommended; see e.g., the recent book by Bishop and Bishop [BiB24], and the references quoted therein.

aims to minimize in some sense the size of the errors $J(x^s) - \tilde{J}(x^s, \hat{r})$, $s = 1, \ldots, q$.

The methodological ideas for parametric cost approximation can also be used for approximation of a target policy μ with a policy from a parametric class $\tilde{\mu}(x, r)$. The training data may be obtained, for example, from rollout control calculations, thus enabling the construction of both value and policy networks that can be combined for use in a perpetual rollout scheme. However, there is an important difference: the approximate cost values $\tilde{J}(x, r)$ are real numbers, whereas the approximate policy values $\tilde{\mu}(x, r)$ are elements of a control space U. Thus if U consists of m dimensional vectors, $\tilde{\mu}(x, r)$ consists of m numerical components. In this case the parametric approximation problems for cost functions and for policies are fairly similar, and both involve continuous space approximations.

On the other hand, the case where the control space is finite, $U = \{u^1, \ldots, u^m\}$. is markedly different. In this case, for any x, $\tilde{\mu}(x, r)$ consists of one of the m possible controls $u^1, \ldots, u^m$. This ushers a connection with traditional classification schemes, whereby objects x are classified as belonging to one of the categories $u^1, \ldots, u^m$, so that $\mu(x)$ *defines the category of* x*, and can be viewed as a classifier*. Some of the most prominent classification schemes actually produce randomized outcomes, i.e., x is associated with a probability distribution

$$\{\tilde{\mu}(u^1, r), \ldots, \tilde{\mu}(u^m, r)\}, \tag{1.25}$$

which is a randomized policy in our policy approximation context; see Fig. 1.3.6. This is done usually for reasons of algorithmic convenience, since many optimization methods, including least squares regression, require that the optimization variables are continuous. In this case, the randomized policy (1.25) can be converted to a nonrandomized policy using a maximization operation: associate x with the control of maximum probability (cf. Fig. 1.3.6),

$$\tilde{\mu}(x, r) \in \arg \max_{i=1,\ldots,m} \tilde{\mu}^i(x, r). \tag{1.26}$$

The use of classification methods for approximation in policy space will be discussed in Chapter 3 (Section 3.4).

1.4 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. We will deal primarily with infinite horizon stochastic problems, where we aim to minimize the total cost over an infinite number of stages, given by

$$J_\pi(x_0) = \lim_{N \to \infty} \underset{\substack{w_k \\ k=0,1,\ldots}}{E} \left\{ \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k), w_k\big) \right\}; \tag{1.27}$$

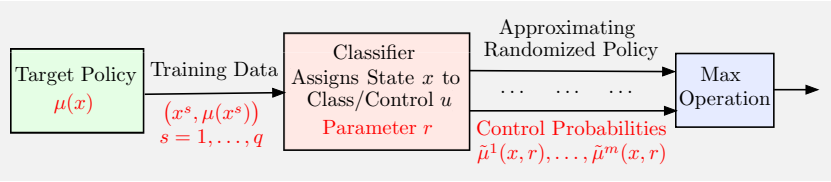

Figure 1.3.6 A general structure for parametric policy approximation for the case where the control space is finite, $U = \{u^1, \ldots, u^m\}$, and its relation to a classification scheme. It produces a randomized policy of the form (1.25), which is converted to a nonrandomized policy through the maximization operation (1.26).

see Fig. 1.4.1. 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 $(0, 1]$. The functions g and f that define the cost per stage and the system equation

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

do not change from one stage to the next. The stochastic disturbances, $w_0, w_1, \ldots$, have a common probability distribution $P(\cdot \mid x_k, u_k)$.

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 three types of problems that we will focus on 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, which we will discuss selectively, arises when termination can be guaranteed only for a subset of policies, which includes all optimal policies. Some common types of SSP 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

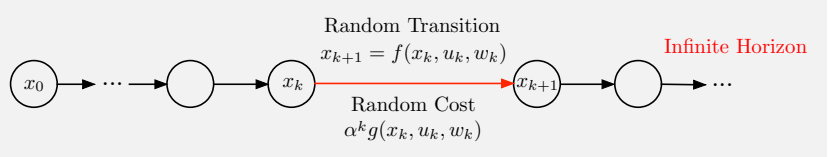

Figure 1.4.1 Illustration of an infinite horizon problem. The system and cost per stage are stationary, except for the use of a discount factor α. If $\alpha = 1$, there is typically a special cost-free termination state that we aim to reach.

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; the DP textbook [Ber17a] provides a detailed account of this conversion, and an accessible introduction to discounted and SSP problems with a finite number of states.

(c) *Deterministic nonnegative cost problems*. Here, the disturbance w_k takes a single known value. Equivalently, there is no disturbance in the system equation and the cost expression, which now take the form

$$x_{k+1} = f(x_k, u_k), \qquad k = 0, 1, \ldots, \tag{1.28}$$

and

$$J_\pi(x_0) = \lim_{N \to \infty} \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k)\big). \tag{1.29}$$

We assume further that there is a cost-free and absorbing termination state t, and that we have

$$g(x, u) \geq 0, \qquad \text{for all } x \neq t,\ u \in U(x), \tag{1.30}$$

and $g(t, u) = 0$ for all $u \in U(t)$. This type of structure expresses the objective to reach or approach t at minimum cost, a classical control problem. An extensive analysis of the undiscounted version of this problem was given in the author's paper [Ber17b].

Discounted stochastic problems with a finite number of states [also referred to as *discounted MDP (abbreviation for Markovian Decision Problem)*] are very common in the DP/RL literature, particularly because of their benign analytical and computational nature. Moreover, there is a widespread belief that discounted MDP can be used as a universal model, i.e., that in practice any other kind of problem (e.g., undiscounted problems with a termination state and/or a continuous state space) can be painlessly converted to a discounted MDP with a discount factor that is close enough to 1. This is questionable, however, for a number of reasons:

(a) Deterministic models are common as well as natural in many practical contexts (including discrete optimization/integer programming problems), so to convert them to MDP does not make sense.

(b) The conversion of a continuous-state problem to a finite-state problem through some kind of discretization involves mathematical subtleties that can lead to serious practical/algorithmic complications. In particular, the character of the optimal solution may be seriously distorted by converting to a discounted MDP through some form of discretization, regardless of how fine the discretization is.

(c) For some practical shortest path contexts it is essential that the termination state is ultimately reached. However, when a discount factor α is introduced in such a problem, the character of the problem may be fundamentally altered. In particular, the threshold for an appropriate value of α may be very close to 1 and may be unknown in practice. For a simple example consider a shortest path problem with states 1 and 2 plus a termination state t. From state 1 we can go to state 2 at cost 0, from state 2 we can go to either state 1 at a small cost $\epsilon > 0$ or to the termination state at a substantial cost $C > 0$. The optimal policy over an infinite horizon is to go from 1 to 2 and from 2 to t. Suppose now that we approximate the problem by introducing a discount factor $\alpha \in (0, 1)$. Then it can be shown that if $\alpha < 1 - \epsilon/C$, it is optimal to move indefinitely around the cycle $1 \to 2 \to 1 \to 2$ and never reach t, while for $\alpha > 1 - \epsilon/C$ the shortest path $2 \to 1 \to t$ will be obtained. Thus the solution of the discounted problem varies discontinuously with α: it changes radically at some threshold, which in general may be unknown.

An important class of problems that we will consider in some detail in this book is finite-state deterministic problems with a large number of states. Finite horizon versions of these problems include challenging discrete optimization problems, whose exact solution is practically impossible. An important fact to keep in mind is that we can transform such problems to infinite horizon SSP problems with a termination state at the end of the horizon, so that the conceptual framework of the present section applies. The approximate solution of discrete optimization problems by RL methods, and particularly by rollout, will be considered in Chapter 2, and has been discussed at length in the books [Ber19a] and [Ber20a].

1.4.1 Infinite Horizon Methodology

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, 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 algorithm

$$J_{k+1}(x) = \min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J_k\big(f(x,u,w)\big)\Big\}, \qquad k = 0,1,\ldots, \tag{1.31}$$

starting from $J_0(x) \equiv 0$.† The algorithm (1.31) 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:

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

$$J^*(x) = \lim_{N\to\infty} J_N(x) \tag{1.32}$$

for all states x.

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

$$J^*(x) = \min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J^*\big(f(x,u,w)\big)\Big\}. \tag{1.33}$$

This is obtained by taking the limit as $N \to \infty$ in the VI algorithm (1.31) using Eq. (1.32). 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.

(c) If $\mu(x)$ attains the minimum in the right-hand side of the Bellman equation (1.33) for each x, then the policy $\{\mu,\mu,\ldots\}$ should be optimal. This type of policy is called *stationary*, and for simplicity it is denoted by μ.

(d) The cost function J_μ of a stationary policy μ satisfies

$$J_\mu(x) = E_w\Big\{g\big(x,\mu(x),w\big) + \alpha J_\mu\big(f(x,\mu(x),w)\big)\Big\}, \qquad \text{for all } x. \tag{1.34}$$

† This is just the finite horizon DP algorithm of Section 1.3.1, except that we have reversed the time indexing to suit our infinite horizon context. 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 for all x,

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

By defining $J_k(x) = V_{N-k}(x)/\alpha^{N-k}$, we obtain the VI algorithm (1.31).

We can view this as just the Bellman equation (1.33) for a different problem, where for each x, the control constraint set $U(x)$ consists of just one control, namely $\mu(x)$. Moreover, we expect that J_μ is obtained in the limit by the VI algorithm:

$$J_\mu(x) = \lim_{N\to\infty} J_{\mu,N}(x), \qquad \text{for all } x,$$

where $J_{\mu,N}$ is the N-stage cost function of μ generated by

$$J_{\mu,k+1}(x) = E_w\Big\{g\big(x,\mu(x),w\big) + \alpha J_{\mu,k}\big(f(x,\mu(x),w)\big)\Big\}, \qquad (1.35)$$

starting from $J_{\mu,0}(x) \equiv 0$ or some other initial condition; cf. Eqs. (1.31)-(1.32).

All four of the preceding results can be shown to hold for finite-state discounted problems, and also for finite-state SSP problems under reasonable assumptions. The results also hold for infinite-state discounted problems, provided the cost per stage function g is bounded over the set of possible values of (x, u, w), in which case we additionally can show that J^* is the unique solution of Bellman's equation. The VI algorithm is also valid under these conditions, 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^*. The associated mathematical proofs can be found in several sources, e.g., [Ber12], Chapter 1, or [Ber19a], Chapter 4.†

It is important to note that for infinite horizon problems, there are additional important algorithms that are amenable to approximation in value space. Approximate policy iteration, Q-learning, temporal difference methods, linear programming, and their variants are some of these; see the RL books [Ber19a], [Ber20a]. For this reason, in the infinite horizon case, there is a richer set of algorithmic options for approximation in value space, despite the fact that the associated mathematical theory is more complex. In this book, we will only discuss approximate forms and variations of the policy iteration algorithm, which we describe next.

Policy Iteration

A major infinite horizon algorithm is *policy iteration* (PI for short). We will argue that PI, together with its variations, forms the foundation for

† For undiscounted problems and discounted problems with unbounded cost per stage, we may still adopt the four preceding results as a working hypothesis. However, we should also be aware that exceptional behavior is possible under unfavorable circumstances, including nonuniqueness of solution of Bellman's equation, and nonconvergence of the VI algorithm to J^* from some initial conditions; see the books [Ber12], [Ber22b].

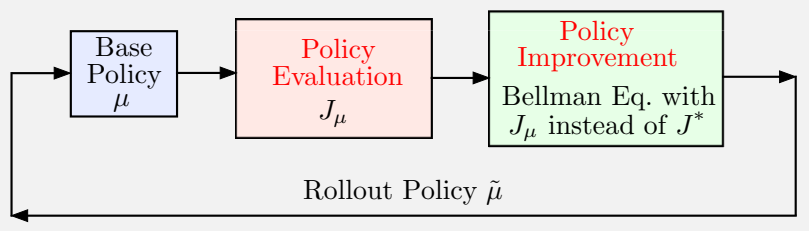

Figure 1.4.2 Schematic illustration of PI as repeated rollout. It generates a sequence of policies, with each policy μ in the sequence being the base policy that generates the next policy $\tilde{\mu}$ in the sequence as the corresponding rollout policy. This rollout policy is used as the base policy in the subsequent iteration.

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. Figure 1.4.2 describes the method as repeated rollout, and indicates that each of its iterations consists of two phases:

(a) *Policy evaluation*, which computes the cost function J_μ of the current (or base) policy μ. One possibility is to solve the corresponding Bellman equation

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

cf. Eq. (1.34). 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.

(b) *Policy improvement*, which computes the "improved" (or rollout) policy $\tilde{\mu}$ using the one-step lookahead minimization

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

We call $\tilde{\mu}$ "improved policy" because we can generally prove that

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

This cost improvement property will be shown in Chapter 2, Section 2.7, and can be used to show that PI produces an optimal policy in a finite number of iterations under favorable conditions (for example for finite-state discounted problems; see the DP books [Ber12], [Ber17a], or the RL book [Ber19a]).

The rollout algorithm in its pure form is just *a single iteration of the PI algorithm*. It starts from a given base policy μ and produces the rollout policy $\tilde{\mu}$. It may be viewed as approximation in value space with one-step lookahead that uses J_μ as terminal cost function approximation.

It has the advantage that it can be applied on-line by computing the needed values of $J_\mu(x)$ by simulation. By contrast, approximate forms of PI for challenging problems, involving for example neural network training, can only be implemented off-line.

1.4.2 Approximation in Value Space - Infinite Horizon

The approximation in value space approach that we discussed in connection with finite horizon problems can be extended in a natural way to infinite horizon problems. Here in place of J^*, we use an approximation $\tilde{J}$, and generate at any state x, a control $\tilde{\mu}(x)$ by the *one-step lookahead minimization*

$$\tilde{\mu}(x) \in \arg\min_{u \in U(x)} E\Big\{ g(x,u,w) + \alpha \tilde{J}\big(f(x,u,w)\big) \Big\}. \tag{1.36}$$

This minimization yields a stationary policy $\{\tilde{\mu}, \tilde{\mu}, \ldots\}$, with cost function denoted $J_{\tilde{\mu}}$ [i.e., $J_{\tilde{\mu}}(x)$ is the total infinite horizon discounted cost obtained when using $\tilde{\mu}$ starting at state x]; see Fig. 1.4.3. Note that when $\tilde{J} = J^*$, the one-step lookahead policy attains the minimum in the Bellman equation (1.33) and is expected to be optimal. This suggests that one should try to use $\tilde{J}$ as close as possible to J^*, which is generally true as we will argue later.

Naturally an important goal to strive for is that $J_{\tilde{\mu}}$ is close to J^* in some sense. However, for classical control problems, which involve steering and maintaining the state near a desired reference state (e.g., problems with a cost-free and absorbing terminal state, and positive cost for all other states), *stability of $\tilde{\mu}$ may be a principal objective*. In this book, we will discuss stability issues primarily for this one class of problems, and *we will consider the policy $\tilde{\mu}$ to be stable if $J_{\tilde{\mu}}$ is real-valued*, i.e.,

$$J_{\tilde{\mu}}(x) < \infty, \qquad \text{for all states } x.$$

Selecting $\tilde{J}$ so that $\tilde{\mu}$ is stable is a question of major interest for some application contexts, such as model predictive and adaptive control, and will be discussed in the next section within the limited context of linear quadratic problems.

ℓ-Step Lookahead

An important extension of one-step lookahead minimization is *ℓ-step lookahead*, whereby at a state x_k we minimize the cost of the first $\ell > 1$ stages with the future costs approximated by a function $\tilde{J}$ (see the bottom half

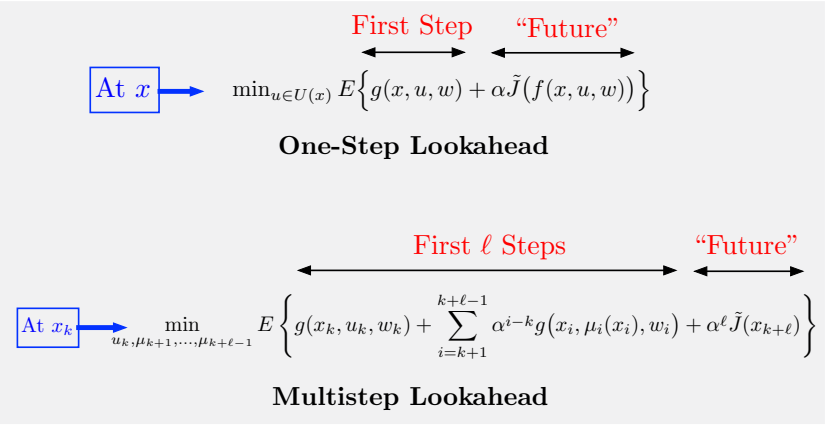

Figure 1.4.3 Schematic illustration of approximation in value space with one-step and ℓ-step lookahead minimization for infinite horizon problems. In the former case, the minimization yields at state x a control $\tilde{u}$, which defines the one-step lookahead policy $\tilde{\mu}$ via

$$\tilde{\mu}(x) = \tilde{u}.$$

In the latter case, the minimization yields a control $\tilde{u}_k$ policies $\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}$. The control $\tilde{u}_k$ is applied at x_k while the remaining sequence $\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}$ is discarded. The control $\tilde{u}_k$ defines the ℓ-step lookahead policy $\tilde{\mu}$.

of Fig. 1.4.3).† This minimization yields a control $\tilde{u}_k$ and a sequence $\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}$. The control $\tilde{u}_k$ is applied at x_k, and defines the ℓ-step lookahead policy $\tilde{\mu}$ via $\tilde{\mu}(x_k) = \tilde{u}_k$, while $\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}$ are discarded. Actually, we may view ℓ-step lookahead minimization as the special case of its one-step counterpart where the lookahead function is the optimal cost function of an $(\ell-1)$-stage DP problem with a terminal cost $\tilde{J}(x_{k+\ell})$ on the state $x_{k+\ell}$ obtained after $\ell-1$ stages.

The motivation for ℓ-step lookahead minimization is that *by increasing the value of ℓ, we may require a less accurate approximation $\tilde{J}$ to obtain good performance*. Otherwise expressed, for the same quality of cost function approximation, better performance may be obtained as ℓ becomes larger. This will be explained visually later, using the formalism of Newton's method in Section 1.5. In particular, for AlphaZero chess, long multistep lookahead is critical for good on-line performance. Another motivation for multistep lookahead is to *enhance the stability properties of the generated on-line policy*, as we will discuss later in Section 1.5. On the other

† On-line play with multistep lookahead minimization (and possibly truncated rollout) is referred to by a number of different names in the RL literature, such as *on-line search*, *predictive learning*, *learning from prediction*, etc; in the model predictive control literature the combined interval of lookahead minimization and truncated rollout is referred as the *prediction interval*.

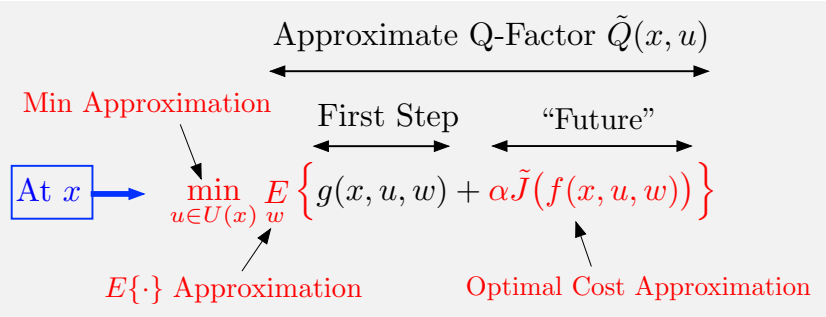

Figure 1.4.4 Approximation in value space with one-step lookahead for infinite horizon problems. There are three potential areas of approximation, which can be considered independently of each other: optimal cost approximation, expected value approximation, and minimization approximation.

hand, solving the multistep lookahead minimization problem, instead of the one-step lookahead counterpart of Eq. (1.36), is more time consuming.

The Three Approximations: Optimal Cost, Expected Value, and Lookahead Minimization Approximations

There are three potential areas of approximation for infinite horizon problems: optimal cost approximation, expected value approximation, and minimization approximation; cf. Fig. 1.4.4. They are similar to their finite horizon counterparts that we discussed in Section 1.3.2. In particular, we have potentially:

(a) A *terminal cost approximation* $\tilde{J}$ *of the optimal cost function* J^*: A major advantage of the infinite horizon context is that only one approximate cost function $\tilde{J}$ is needed, rather than the N functions $\tilde{J}_1,\ldots,\tilde{J}_N$ of the N-step horizon case.

(b) An *approximation of the expected value operation*: This operation can be very time consuming. It may be simplified in various ways. For example some of the random quantities $w_k, w_{k+1},\ldots,w_{k+\ell-1}$ appearing in the ℓ-step lookahead minimization may be replaced by deterministic quantities; this is another example of the *certainty equivalence approach*, which we discussed in Section 1.3.2.

(c) A *simplification of the minimization operation*: For example in multiagent problems the control consists of multiple components,

$$u=(u^1,\ldots,u^m),$$

with each component u^i chosen by a different agent/decision maker. In this case the size of the control space can be enormous, but it

can be simplified in ways that will be discussed later (e.g., choosing components sequentially, one-agent-at-a-time). This will form the core of our approach to multiagent problems; see Section 1.6.5 and Chapter 2, Section 2.9.

We will next describe briefly various approaches for selecting the terminal cost function approximation.

Constructing Terminal Cost Approximations for On-Line Play

A major issue in value space approximation is the construction of a suitable approximate cost function $\tilde{J}$. This can be done in many different ways, giving rise to some of the principal RL methods.

For example, $\tilde{J}$ may be constructed with sophisticated off-line training methods. Alternatively, the approximate values $\tilde{J}(x)$ may be obtained on-line as needed with truncated rollout, by running an off-line obtained policy for a suitably large number of steps, starting from x, and supplementing it with a suitable, perhaps primitive, terminal cost approximation.

For orientation purposes, let us describe briefly four broad types of approximation. We will return to these approaches later, and we also refer to the RL and approximate DP literature for more detailed discussions.

(a) *Off-line problem approximation*: Here the function $\tilde{J}$ is computed off-line as the optimal or nearly optimal cost function of a simplified optimization problem, which is more convenient for computation. Simplifications may include exploiting decomposable structure, reducing the size of the state space, neglecting some of the constraints, and ignoring various types of uncertainties. For example we may consider using as $\tilde{J}$ the cost function of a related deterministic problem, obtained through some form of certainty equivalence approximation, thus allowing computation of $\tilde{J}$ by gradient-based optimal control methods or shortest path-type methods.
A major type of problem approximation method is *aggregation*, described in Section 3.6, and in the books [Ber12], [Ber19a] and papers [Ber18a], [Ber18b]. Aggregation provides a systematic procedure to simplify a given problem by grouping states together into a relatively small number of subsets, called aggregate states. The optimal cost function of the simpler aggregate problem is computed by exact DP methods, possibly involving the use of simulation. This cost function is then used to provide an approximation $\tilde{J}$ to the optimal cost function J^* of the original problem, using some form of interpolation.

(b) *On-line simulation*: This possibility arises in rollout algorithms for stochastic problems, where we use Monte-Carlo simulation and some suboptimal policy μ (the base policy) to compute (whenever needed) values $\tilde{J}(x)$ that are exactly or approximately equal to $J_\mu(x)$. The policy μ may be obtained by any method, e.g., one based on heuris-

tic reasoning (such as in the case of the traveling salesman Example 1.2.3), or off-line training based on a more principled approach, such as approximate policy iteration or approximation in policy space. Note that while simulation is time-consuming, it is uniquely well-suited for the use of parallel computation. Moreover, it can be simplified through the use of certainty equivalence approximations.

(c) *On-line approximate optimization*. This approach involves the solution of a suitably constructed shorter horizon version of the problem, with a simple terminal cost approximation. It can be viewed as either approximation in value space with multistep lookahead, or as a form of rollout algorithm. It is often used in model predictive control (MPC).

(d) *Parametric cost approximation*, where $\tilde{J}$ is obtained from a given parametric class of functions $J(x, r)$, where r is a parameter vector, selected by a suitable algorithm. The parametric class typically involves prominent characteristics of x called *features*, which can be obtained either through insight into the problem at hand, or by using training data and some form of neural network (see Chapter 3).

Such methods include approximate forms of PI, as discussed in Section 1.1 in connection with chess and backgammon. The policy evaluation portion of the PI algorithm can be done by approximating the cost function of the current policy using an approximation architecture such as a neural network (see Chapter 3). It can also be done with stochastic iterative algorithms such as TD(λ), LSPE(λ), and LSTD(λ), which are described in the DP book [Ber12] and the RL book [Ber19a]. These methods are somewhat peripheral to our course, and will not be discussed at any length. We note, however, that approximate PI methods do not just yield a parametric approximate cost function $J(x, r)$, but also a suboptimal policy, which can be improved on-line by using (possibly truncated) rollout.

Aside from approximate PI, parametric approximate cost functions $J(x, r)$ may be obtained off-line with methods such as Q-learning, linear programming, and aggregation methods, which are also discussed in the books [Ber12] and [Ber19a].

Let us also mention that for problems with special structure, $\tilde{J}$ may be chosen so that the one-step lookahead minimization (1.36) is facilitated. In fact, under favorable circumstances, the lookahead minimization may be carried out in closed form. An example is when the system is nonlinear, but the control enters linearly in the system equation and quadratically in the cost function, while the terminal cost approximation is quadratic. Then the one-step lookahead minimization can be carried out analytically, because it involves a function that is quadratic in u.

From Off-Line Training to On-Line Play

Generally off-line training will produce either just a cost approximation (as in the case of TD-Gammon), or just a policy (as for example by some approximation in policy space/policy gradient approach), or both (as in the case of AlphaZero). We have already discussed in this section one-step lookahead and multistep lookahead schemes to implement on-line approximation in value space using $\tilde{J}$; cf. Fig. 1.4.3. Let us now consider some additional possibilities, which involve the use of a policy μ that has been obtained off-line (possibly in addition to a terminal cost approximation). Here are some of the main possibilities:

(a) *Given a policy* μ that has been obtained off-line, we may use as terminal cost approximation $\tilde{J}$ the cost function J_μ of the policy. For the case of one-step lookahead, this requires a policy evaluation operation, and can be done on-line, by computing (possibly by simulation) just the values of

$$E\Big\{J_\mu\big(f(x_k,u_k,w_k)\big)\Big\}$$

that are needed [cf. Eq. (1.36)]. For the case of ℓ-step lookahead, the values

$$E\big\{J_\mu(x_{k+\ell})\big\}$$

for all states $x_{k+\ell}$ that are reachable in ℓ steps starting from x_k are needed. This is the simplest form of rollout, and only requires the off-line construction of the policy μ.

(b) *Given a terminal cost approximation* $\tilde{J}$ that has been obtained off-line, we may use it on-line to compute fast when needed the controls of a corresponding one-step or multistep lookahead policy $\tilde{\mu}$. The policy $\tilde{\mu}$ can in turn be used for rollout as in (a) above. In a truncated variation of this scheme, we may also use $\tilde{J}$ to approximate the tail end of the rollout process (an example of this is the rollout-based TD-Gammon algorithm).

(c) *Given a policy* μ *and a terminal cost approximation* $\tilde{J}$, we may use them together in a truncated rollout scheme, whereby the tail end of the rollout with μ is approximated using the cost approximation $\tilde{J}$. This is similar to the truncated rollout scheme noted in (b) above, except that the policy μ is computed off-line rather than on-line using $\tilde{J}$ and one-step or multistep lookahead.

The preceding three possibilities are the principal ones for using the results of off-line training within on-line play schemes. Naturally, there are variations where additional information is computed off-line to facilitate and/or expedite the on-line play algorithm. As an example, in MPC, in addition to a terminal cost approximation, a target tube may need to be computed off-line in order to guarantee that some state constraints can

be satisfied on-line; see the discussion of MPC in Section 1.6.7. Other examples of this type will be noted in the context of specific applications.

Finally, let us note that while we have emphasized approximation in value space with cost function approximation, our discussion applies to Q-factor approximation, involving functions

$$\tilde{Q}(x,u) \approx E\Big\{g(x,u,w) + \alpha J^*\big(f(x,u,w)\big)\Big\}.$$

The corresponding one-step lookahead scheme has the form

$$\tilde{\mu}(x) \in \arg\min_{u\in U(x)} E\Big\{g(x,u,w)+\alpha \min_{u'\in U(f(x,u,w))} \tilde{Q}\big(f(x,u,w),u'\big)\Big\}; \quad (1.37)$$

cf. Eq. (1.36). The second term on the right in the above equation represents the cost function approximation

$$\tilde{J}\big(f(x,u,w)\big) = \min_{u'\in U(f(x,u,w))} \tilde{Q}\big(f(x,u,w),u'\big).$$

The use of Q-factors is common in the "model-free" case where a computer simulator is used to generate samples of w, and corresponding values of g and f. Then, having obtained $\tilde{Q}$ through off-line training, the one-step lookahead minimization in Eq. (1.37) must be performed on-line with the use of the simulator.

1.4.3 Understanding Approximation in Value Space

We will now discuss some of our aims as we try to get insight into the process of approximation in value space. Clearly, it makes sense to approximate J^* with a function $\tilde{J}$ that is as close as possible to J^*. However, we should also try to understand quantitatively the relation between $\tilde{J}$ and $J_{\tilde{\mu}}$, the cost function of the resulting one-step lookahead (or multistep lookahead) policy $\tilde{\mu}$. Interesting questions in this regard are the following:

(a) *How is the quality of the lookahead policy $\tilde{\mu}$ affected by the quality of the off-line training?* A related question is how much should we care about improving $\tilde{J}$ through a longer and more sophisticated training process, for a given approximation architecture? A fundamental fact that provides a lot of insight in this respect is that *$J_{\tilde{\mu}}$ is the result of a step of Newton's method that starts at $\tilde{J}$ and is applied to the Bellman Eq. (1.33).* This will be the focus of our discussion in the next section, and has been a major point in the narrative of the author's books, [Ber20a] and [Ber22a].

A related fact is that in approximation in value space with multistep lookahead, *$J_{\tilde{\mu}}$ is the result of a step of Newton's method that starts at the function obtained by applying multiple value iterations to $\tilde{J}$.*

(b) *How do simplifications in the multistep lookahead implementation affect $J_{\tilde{\mu}}$?* The Newton step interpretation of approximation in value space leads to an important insight into the special character of the initial step of the multistep lookahead. In particular, *it is only the first step that acts as the Newton step, and needs to be implemented with precision*. The subsequent steps are value iterations, which only serve to enhance the quality of the starting point of the Newton step, and hence *their precise implementation is not critical*.

This idea suggests that simplifications of the lookahead steps after the first can be implemented with relatively small (if any) performance loss for the multistep lookahead policy. Important examples of such simplifications are the use of certainty equivalence (Sections 1.6.7, 2.7.2, 2.8.3), and forms of pruning of the lookahead tree (Section 2.4). In practical terms, simplifications after the first step of the multistep lookahead can save a lot of on-line computation, which can be fruitfully invested in extending the length of the lookahead.

(c) *When is $\tilde{\mu}$ stable?* The question of stability is very important in many control applications where the objective is to keep the state near some reference point or trajectory. Indeed, in such applications, stability is the dominant concern, and optimality is secondary by comparison. Among others, here we are interested to characterize the set of terminal cost approximations $\tilde{J}$ that lead to a stable $\tilde{\mu}$.

(d) *How does the length of lookahead minimization or the length of the truncated rollout affect the stability and quality of the multistep lookahead policy $\tilde{\mu}$?* While it is generally true that the length of lookahead has a beneficial effect on quality, it turns out that it also has a beneficial effect on the stability properties of the multistep lookahead policy, and we are interested in the mechanism by which this occurs.

In what follows we will be keeping in mind these questions. In particular, in the next section, we will discuss them in the context of the simple and convenient linear quadratic problem. Our conclusions, however, hold within a far more general context with the aid of the abstract DP formalism; see the author's books [Ber20a] and [Ber22a] for a broader presentation and analysis, which address these questions in greater detail and generality.

1.5 NEWTON'S METHOD - LINEAR QUADRATIC PROBLEMS

We will now aim to understand the character of the Bellman equation, approximation in value space, and the VI and PI algorithms within the context of an important deterministic problem. This is the classical continuous-spaces problem where the system is linear, with no control constraints, and the cost function is nonnegative quadratic. While this prob-

lem can be solved analytically, it provides a uniquely insightful context for understanding visually the Bellman equation and its algorithmic solution, both exactly and approximately.

In its general form, the problem deals with the system

$$x_{k+1} = Ax_k + Bu_k,$$

where x_k and u_k are elements of the Euclidean spaces $\Re^n$ and $\Re^m$, respectively, A is an $n \times n$ matrix, and B is an $n \times m$ matrix. It is assumed that there are no control constraints. The cost per stage is quadratic of the form

$$g(x, u) = x'Qx + u'Ru,$$

where Q and R are positive definite symmetric matrices of dimensions $n \times n$ and $m \times m$, respectively (all finite-dimensional vectors in this work are viewed as column vectors, and a prime denotes transposition). The analysis of this problem is well known and is given with proofs in several control theory texts, including the author's DP books [Ber17a] and [Ber12].

In what follows, we will focus for simplicity only on the one-dimensional version of the problem, where the system has the form

$$x_{k+1} = ax_k + bu_k; \tag{1.38}$$

cf. Example 1.3.1. Here the state x_k and the control u_k are scalars, and the coefficients a and b are also scalars, with $b \neq 0$. The cost function is undiscounted and has the form

$$\sum_{k=0}^{\infty}(qx_k^2 + ru_k^2), \tag{1.39}$$

where q and r are positive scalars. The one-dimensional case allows a convenient and insightful analysis of the algorithmic issues that are central for our purposes. This analysis generalizes to multidimensional linear quadratic problems and beyond, but requires a more demanding mathematical treatment.

The Riccati Equation and its Justification

The analytical results for our problem may be obtained by taking the limit in the results derived in the finite horizon Example 1.3.1, as the horizon length tends to infinity. In particular, we can show that the optimal cost function is expected to be quadratic of the form

$$J^*(x) = K^*x^2, \tag{1.40}$$

where the scalar K^* solves the equation

$$K = F(K), \tag{1.41}$$

with F defined by

$$F(K) = \frac{a^2 rK}{r + b^2 K} + q. \tag{1.42}$$

This is the limiting form of Eq. (1.19).

Moreover, the optimal policy is linear of the form

$$\mu^*(x) = L^* x, \tag{1.43}$$

where L^* is the scalar given by

$$L^* = -\frac{abK^*}{r + b^2 K^*}. \tag{1.44}$$

To justify Eqs. (1.41)-(1.44), we show that J^* as given by Eq. (1.40), satisfies the Bellman equation

$$J(x) = \min_{u \in \Re} \big\{ qx^2 + ru^2 + J(ax + bu) \big\}, \tag{1.45}$$

and that $\mu^*(x)$, as given by Eqs. (1.43)-(1.44), attains the minimum above for every x when $J = J^*$. Indeed for any quadratic cost function $J(x) = Kx^2$ with $K \geq 0$, the minimization in Bellman's equation (1.45) is written as

$$\min_{u \in \Re} \big\{ qx^2 + ru^2 + K(ax + bu)^2 \big\}. \tag{1.46}$$

Thus it involves minimization of a positive definite quadratic in u and can be done analytically. By setting to 0 the derivative with respect to u of the expression in braces in Eq. (1.46), we obtain

$$0 = 2ru + 2bK(ax + bu),$$

so the minimizing control and corresponding policy are given by

$$\mu_K(x) = L_K x, \tag{1.47}$$

where

$$L_K = -\frac{abK}{r + b^2 K}. \tag{1.48}$$

By substituting this control, the minimized expression (1.46) takes the form

$$\Big(q + rL_K^2 + K(a + bL_K)^2 \Big) x^2.$$

After straightforward algebra, using Eq. (1.48) for L_K, it can be verified that this expression is written as $F(K)x^2$, with F given by Eq. (1.42). Thus when $J(x) = Kx^2$, the Bellman equation (1.45) takes the form

$$Kx^2 = F(K)x^2$$

or equivalently $K = F(K)$ [cf. Eq. (1.41)].

In conclusion, when restricted to quadratic functions $J(x) = Kx^2$ with $K \geq 0$, the Bellman equation (1.45) is equivalent to the equation

$$K = F(K) = \frac{a^2 rK}{r + b^2 K} + q. \tag{1.49}$$

We refer to this equation as the *Riccati equation*† and to the function F as the *Riccati operator*.‡ Moreover, the policy corresponding to K^*, as per Eqs. (1.47)-(1.48), attains the minimum in Bellman's equation, and is given by Eqs. (1.43)-(1.44).

The Riccati equation can be visualized and solved graphically as illustrated in Fig. 1.5.1. As shown in the figure, the quadratic coefficient K^* that corresponds to the optimal cost function J^* [cf. Eq. (1.40)] is the unique solution of the Riccati equation $K = F(K)$ within the nonnegative real line.

The Riccati Equation for a Stable Linear Policy

We can also characterize the cost function of a policy μ that is linear of the form $\mu(x) = Lx$, and is also stable, in the sense that the scalar L satisfies $|a + bL| < 1$, so that the corresponding closed-loop system

$$x_{k+1} = (a + bL)x_k$$

is stable (its state x_k converges to 0 as $k \to \infty$). In particular, we can show that its cost function has the form

$$J_\mu(x) = K_L x^2,$$

† This is an algebraic form of the Riccati differential equation, which was invented in its one-dimensional form by count Jacopo Riccati in the 1700s, and has played an important role in control theory. It has been studied extensively in its differential and difference matrix versions; see the book by Lancaster and Rodman [LR95], and the paper collection by Bittanti, Laub, and Willems [BLW91], which also includes a historical account by Bittanti [Bit91] of Riccati's remarkable life and accomplishments.

‡ The Riccati operator is a special case of the *Bellman operator*, denoted by T, which transforms a function J into the right side of Bellman's equation:

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

Thus the Bellman operator T transforms a function J of x into another function TJ also of x. Bellman operators allow a succinct abstract description of the problem's data, and are fundamental in the theory of abstract DP (see the author's monographs [Ber22a] and [Ber22b]). We may view the Riccati operator as the restriction of the Bellman operator to the subspace of quadratic functions of x.

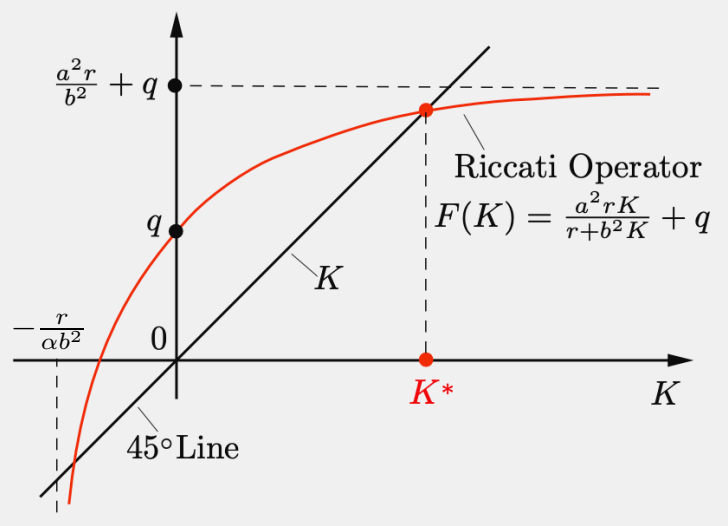

Figure 1.5.1 Graphical construction of the solutions of the Riccati equation (1.41)-(1.42) for the linear quadratic problem. The optimal cost function is $J^*(x) = K^*x^2$, where the scalar K^* solves the fixed point equation $K = F(K)$, with F being the Riccati operator given by

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

Note that F is concave and monotonically increasing in the interval $(-r/b^2, \infty)$ and "flattens out" as $K \to \infty$, as shown in the figure. The quadratic Riccati equation $K = F(K)$ also has another solution, denoted by $\bar{K}$, which is negative and therefore of no interest.

where K_L solves the equation

$$K = F_L(K), \tag{1.50}$$

with F_L defined by

$$F_L(K) = (a + bL)^2K + q + rL^2. \tag{1.51}$$

This equation is called the *Riccati equation for the stable policy* $\mu(x) = Lx$. It is illustrated in Fig. 1.5.2, and it is linear, with linear coefficient $(a+bL)^2$ that is strictly less than 1. Hence the line that represents the graph of F_L intersects the 45-degree line at a unique point, which defines the quadratic cost coefficient K_L.

The Riccati equation (1.50)-(1.51) for $\mu(x) = Lx$ may be justified by verifying that it is in fact the Bellman equation for μ,

$$J(x) = (q + rL^2)x^2 + J\big((a + bL)x\big),$$

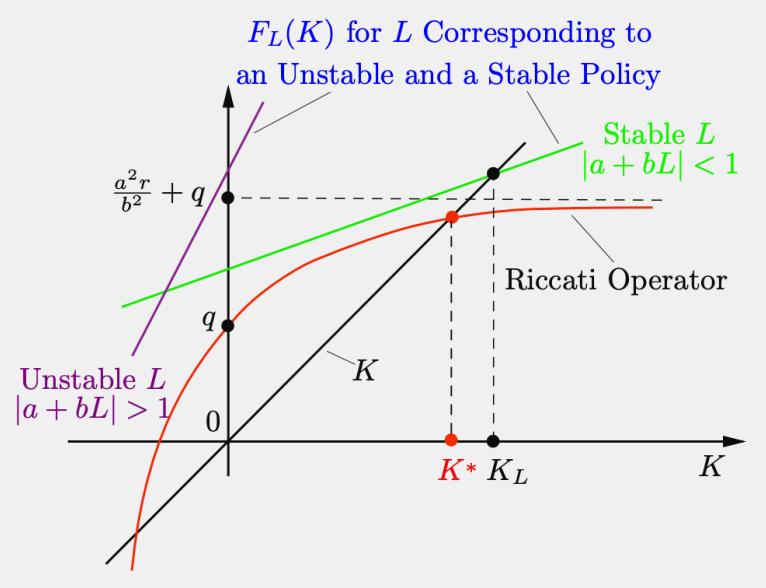

Figure 1.5.2 Illustration of the construction of the cost function of a linear policy $\mu(x) = Lx$, which is stable, i.e., $|a + bL| < 1$. The cost function $J_\mu(x)$ has the form

$$J_\mu(x) = K_L x^2,$$

with K_L obtained as the unique solution of the linear equation $K = F_L(K)$, where

$$F_L(K) = (a + bL)^2 K + q + rL^2,$$

is the Riccati equation operator corresponding to $\mu(x) = Lx$. If μ is not stable, i.e., $|a + bL| \geq 1$, we have $J_\mu(x) = \infty$ for all $x \neq 0$, but the equation has $K = F_L(K)$ still has a solution that is of no interest within our context.

[cf. Eq. (1.34)], restricted to quadratic functions of the form $J(x) = Kx^2$.

We note, however, that $J_\mu(x) = K_L x^2$ is the solution of the Riccati equation (1.50)-(1.51) only when $\mu(x) = Lx$ is stable. If μ is not stable, i.e., $|a + bL| \geq 1$, then (since $q > 0$ and $r > 0$) we have $J_\mu(x) = \infty$ for all $x \neq 0$. Then, the Riccati equation (1.50)-(1.51) is still defined, but its solution is negative and is of no interest within our context.

Value Iteration

The VI algorithm for our linear quadratic problem is given by

$$J_{k+1}(x) = \min_{u \in \Re} \big\{ qx^2 + ru^2 + J_k(ax + bu) \big\}.$$

When J_k is quadratic of the form $J_k(x) = K_k x^2$ with $K_k \geq 0$, it can be seen that the VI iterate J_{k+1} is also quadratic of the form $J_{k+1}(x) = K_{k+1} x^2$,

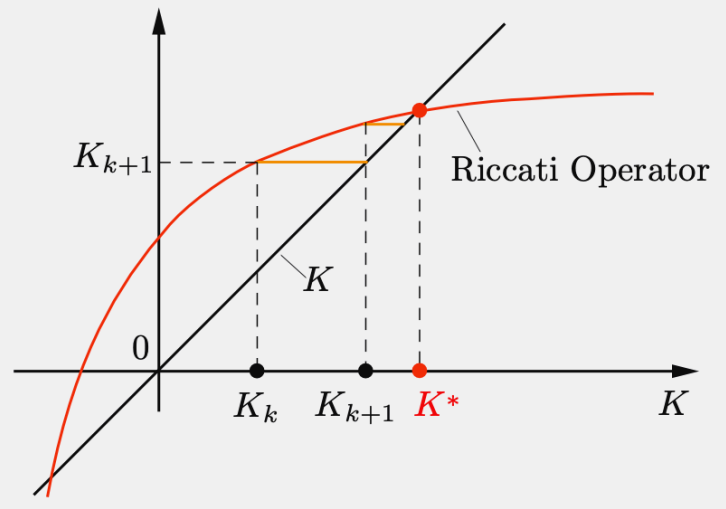

Figure 1.5.3 Graphical illustration of value iteration for the linear quadratic problem. It has the form $K_{k+1} = F(K_k)$, where F is the Riccati operator,

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

The algorithm converges to K^* starting from any $K_0 \geq 0$.

where

$$K_{k+1} = F(K_k),$$

with F being the Riccati operator of Eq. (1.49). The algorithm is illustrated in Fig. 1.5.3. As can be seen from the figure, when starting from any $K_0 \geq 0$, the algorithm generates a sequence $\{K_k\}$ of nonnegative scalars that converges to K^*.

1.5.1 Visualizing Approximation in Value Space - Region of Stability

The use of Riccati equations allows insightful visualization of approximation in value space. This visualization, although specialized to linear quadratic problems, is consistent with related visualizations for more general infinite horizon problems; this is a recurring theme in what follows. In particular, in the books [Ber20a] and [Ber22a], Bellman operators, which define the Bellman equations, are used in place of Riccati operators, which define the Riccati equations.

In summary, we will aim to show that:

(a) Approximation in value space with one-step lookahead can be viewed as a Newton step for solving the Bellman equation, and maps the

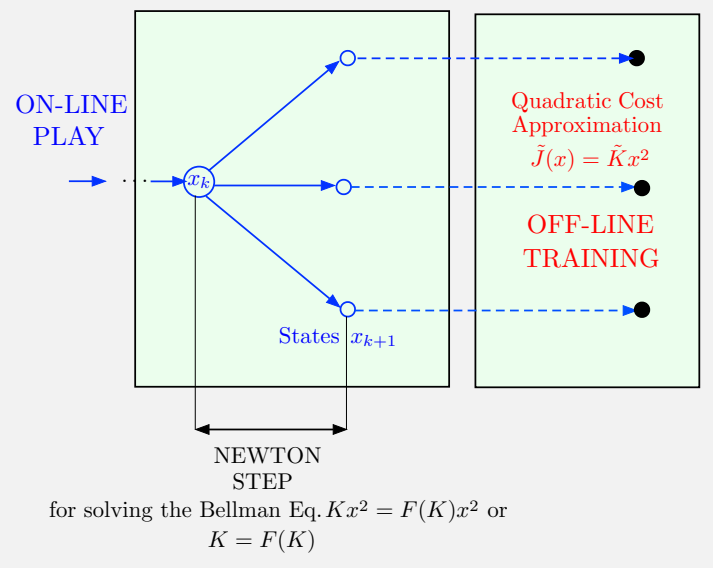

Figure 1.5.4 Illustration of the interpretation of approximation in value space with one-step lookahead as a Newton step that maps $\tilde{J}$ to the cost function $J_{\tilde{\mu}}$ of the one-step lookahead policy.

terminal cost function approximation $\tilde{J}$ to the cost function $J_{\tilde{\mu}}$ of the one-step lookahead policy; see Fig. 1.5.4.

(b) Approximation in value space with multistep lookahead and truncated rollout can be viewed as a Newton step for solving the Bellman equation, and maps the result of multiple VI iterations starting with the terminal cost function approximation $\tilde{J}$ to the cost function $J_{\tilde{\mu}}$ of the multistep lookahead policy; see Fig. 1.5.5.

Our derivation will be given for the one-dimensional linear quadratic problem, but *applies far more generally*. The reason is that the Bellman equation is valid universally in DP, and the corresponding Bellman operator has a concavity property that is well-suited for the application of Newton's method; see the books [Ber20a] and [Ber22a], where the connection of approximation in value space with Newton's method was first developed in detail.

Let us consider one-step lookahead minimization with any terminal cost function approximation of the form $\tilde{J}(x) = Kx^2$, where $K \geq 0$. We have derived the one-step lookahead policy $\mu_K(x)$ in Eqs. (1.47)-(1.48), by minimizing the right side of Bellman's equation when $J(x) = Kx^2$:

$$\min_{u \in \Re} \{qx^2 + ru^2 + K(ax + bu)^2\}.$$

We can break this minimization into a sequence of two minimizations as

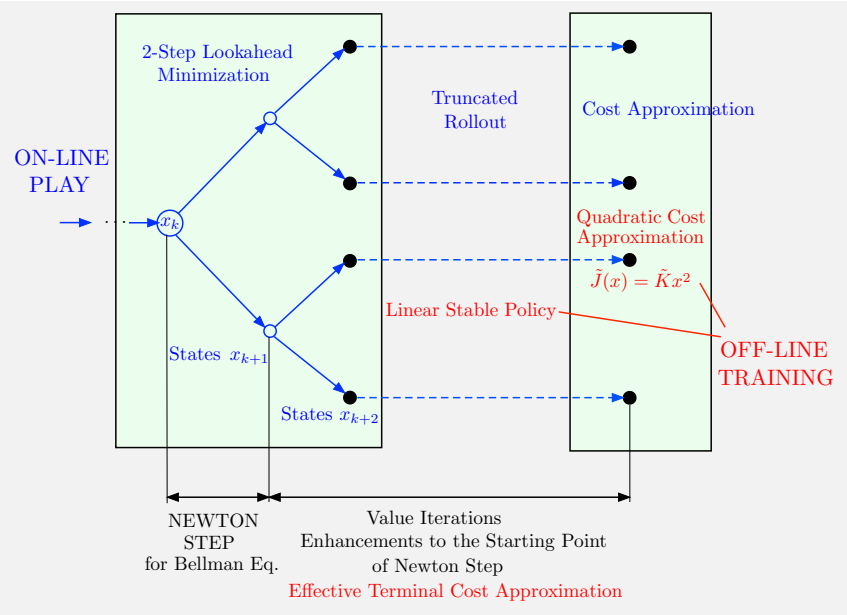

Figure 1.5.5 Illustration of the interpretation of approximation in value space with multistep lookahead and truncated rollout as a Newton step, which maps the result of multiple VI iterations starting with the terminal cost function approximation $\tilde{J}$ to the cost function $J_{\tilde{\mu}}$ of the multistep lookahead policy.

follows:

$$F(K)x^2 = \min_{L\in\Re} \min_{u=Lx} \left\{qx^2+ru^2+K(ax+bu)^2\right\} = \min_{L\in\Re} \left\{q+bL+K(a+bL)^2\right\}x^2.$$

From this equation, it follows that

$$F(K) = \min_{L\in\Re} F_L(K), \tag{1.52}$$

where the function $F_L(K)$ is defined by

$$F_L(K) = (a+bL)^2K + q + bL. \tag{1.53}$$

Figure 1.5.6 illustrates the relation (1.52)-(1.53), and shows how the graph of *the Riccati operator F can be obtained as the lower envelope of the linear operators F_L*, as L ranges over the real numbers.

One-Step Lookahead Minimization and Newton's Method

Let us now fix the terminal cost function approximation to some $\tilde{K}x^2$, where $\tilde{K} \geq 0$, and consider the corresponding one-step lookahead policy,

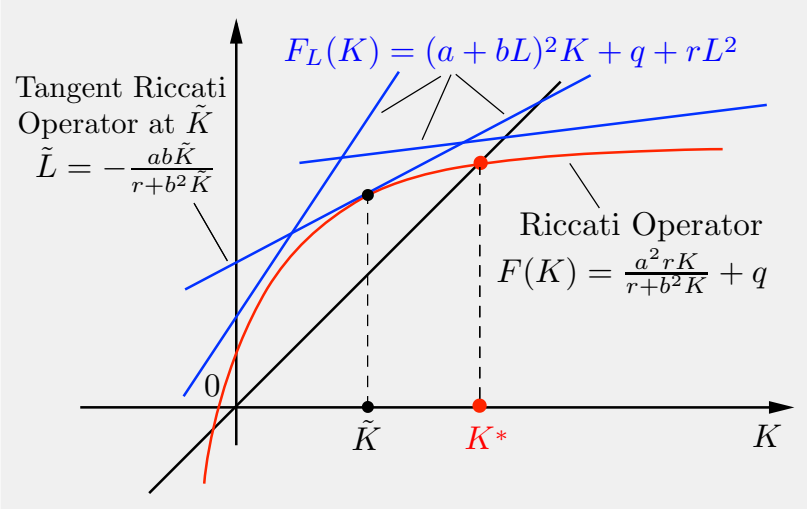

Figure 1.5.6 Illustration of how the graph of the Riccati operator F can be obtained as the lower envelope of the linear operators

$$F_L(K) = (a + bL)^2K + q + bL,$$

as L ranges over the real numbers. We have

$$F(K) = \min_{L\in\Re} F_L(K);$$

cf. Eq. (1.52). Moreover, for any fixed $\tilde{K}$, the scalar $\tilde{L}$ that attains the minimum is given by

$$\tilde{L} = -\frac{ab\tilde{K}}{r + b^2\tilde{K}}$$

[cf. Eq. (1.48)], and is such that the line corresponding to the graph of $F_{\tilde{L}}$ is tangent to the graph of F at $\tilde{K}$, as shown in the figure.

which we will denote by $\tilde{\mu}$. Figure 1.5.7 illustrates the corresponding linear function $F_{\tilde{L}}$, and shows that its graph is a tangent line to the graph of F at the point K [cf. Fig. 1.5.6 and Eq. (1.53)].

Thus the function $F_{\tilde{L}}$ can be viewed as a linearization of F at the point K, and defines a linearized problem: to find a solution of the equation

$$K = F_{\tilde{L}}(K) = q + b\tilde{L}^2 + K(a + b\tilde{L})^2.$$

The important point now is that *the solution of this equation, denoted $K_{\tilde{L}}$, is the same as the one obtained from a single iteration of Newton's method for solving the Riccati equation, starting from the point $\tilde{K}$*. This is illustrated in Fig. 1.5.7, and is also justified analytically in Exercise 1.7.

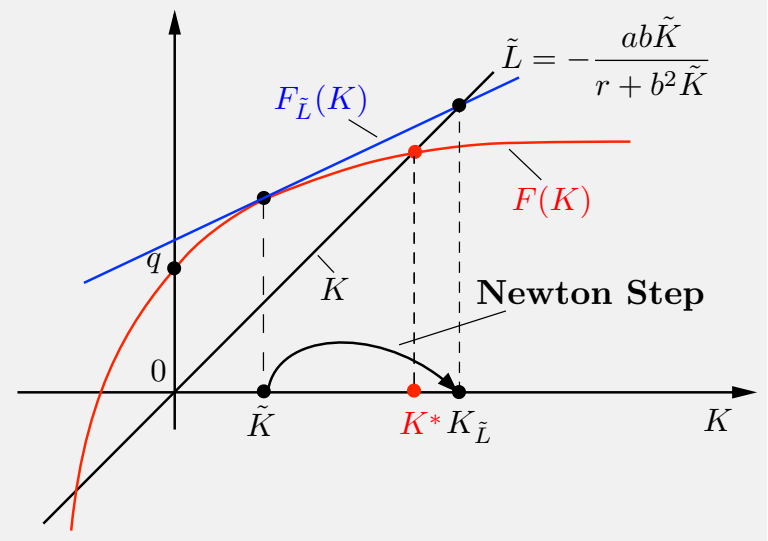

Figure 1.5.7 Illustration of approximation in value space with one-step lookahead for the linear quadratic problem. Given a terminal cost approximation $\tilde{J} = \tilde{K}x^2$, we compute the corresponding linear policy $\tilde{\mu}(x) = \tilde{L}x$, where

$$\tilde{L} = -\frac{ab\tilde{K}}{r + b^2\tilde{K}},$$

and the corresponding cost function $K_{\tilde{L}}x^2$, using the Newton step shown.

To explain this connection, we note that the classical form of 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. Assuming T is differentiable, the linearization is obtained by 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 vector y_k, as indicated in Fig. 1.5.7.

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). There are extensions of Newton's method that are based on solving a linearized

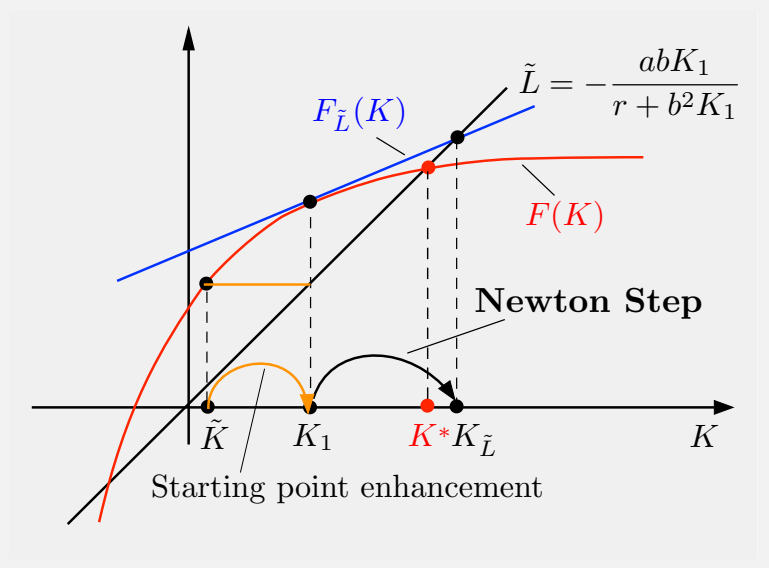

Figure 1.5.8 Illustration of approximation in value space with two-step lookahead for the linear quadratic problem. Starting with a terminal cost approximation $\tilde{J}(x) = \tilde{K}x^2$, we obtain K_1 using a single value iteration, thereby enhancing the starting point of the Newton step. We then compute the corresponding linear policy $\tilde{\mu}(x) = \tilde{L}x$, where

$$\tilde{L} = -\frac{abK_1}{r + b^2K_1}$$

and the corresponding cost function $K_{\tilde{L}}x^2$, using the Newton step shown. The figure shows that for any $K \geq 0$, the corresponding ℓ-step lookahead policy will be stable for all ℓ larger than some threshold.

system at the current iterate, but relax the differentiability requirement to piecewise differentiability, and/or component concavity, while maintaining the either a quadratic or a similarly fast superlinear convergence property of the method; see the monograph [Ber22a] (Appendix A) and the paper [Ber22c], which also provide a convergence analysis.

Note also that if the one-step lookahead policy is stable, i.e., $|a+b\tilde{L}| < 1$, then $K_{\tilde{L}}$ is the quadratic cost coefficient of its cost function, i.e.,

$$J_{\tilde{\mu}}(x) = K_{\tilde{L}}x^2.$$

The reason is that $J_{\tilde{\mu}}$ solves the Bellman equation for policy $\tilde{\mu}$. On the other hand, if $\tilde{\mu}$ is not stable, then in view of the positive definite quadratic cost per stage, we have $J_{\tilde{\mu}}(x) = \infty$ for all $x \neq 0$.

Multistep Lookahead

In the case of ℓ-step lookahead minimization, a similar Newton step inter-

Riccati Equation Formulas for One-Dimensional Problems

Riccati equation for minimization [cf. Eqs. (1.41) and (1.42)]

$$K = F(K), \qquad F(K) = \frac{a^2 rK}{r + b^2 K} + q.$$

Riccati equation for a linear policy $\mu(x) = Lx$

$$K = F_L(K), \qquad F_L(K) = (a + bL)^2 K + q + rL^2.$$

Cost coefficient K_L of a stable linear policy $\mu(x) = Lx$

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

Linear coefficient L_K of the one-step lookahead linear policy μ_K for K in the region of stability [cf. Eq. (1.48)]

$$L_K = \arg\min_L F_L(K) = -\frac{abK}{r + b^2 K}.$$

Quadratic cost coefficient $\tilde{K}$ of a one-step lookahead linear policy μ_K for K in the region of stability

Obtained as the solution of the linearized Riccati equation

$$\tilde{K} = F_{L_K}(\tilde{K}),$$

or equivalently by a Newton iteration starting from K.

pretation is possible. Instead of linearizing F at $\tilde{K}$, we linearize at

$$K_{\ell-1} = F^{\ell-1}(\tilde{K}),$$

i.e., the result of $\ell - 1$ successive applications of F starting with $\tilde{K}$. Each application of F corresponds to a value iteration. Thus *the effective starting point for the Newton step is* $F^{\ell-1}(\tilde{K})$. Figure 1.5.8 depicts the case $\ell = 2$.

Region of Stability

It is also useful to define the *region of stability* as the set of $K \geq 0$ such

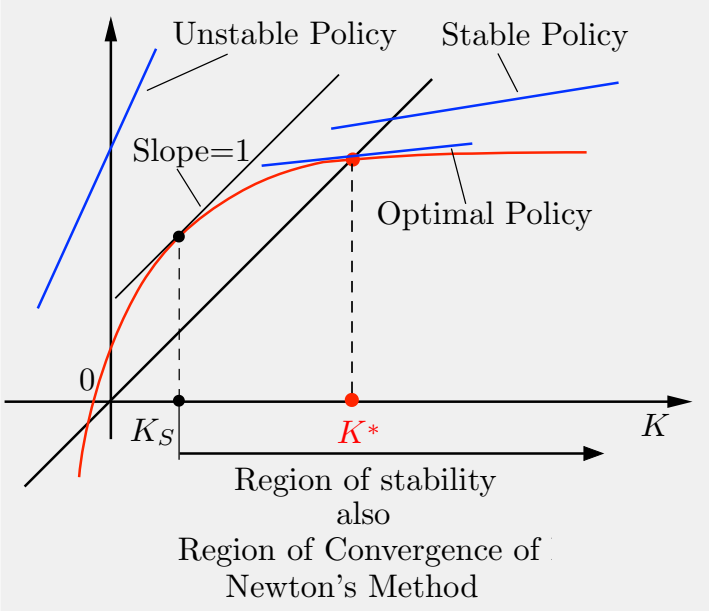

Figure 1.5.9 Illustration of the region of stability, i.e., the set of $K \geq 0$ such that the one-step lookahead policy μ_K is stable. This is also the set of initial conditions for which Newton's method converges to K^* asymptotically.

that

$$|a + bL_K| < 1,$$

where L_K is the linear coefficient of the one-step lookahead policy corresponding to K; cf. Eq. (1.48). The region of stability may also be viewed as *the region of convergence of Newton's method*. It is the set of starting points K for which Newton's method, applied to the Riccati equation $F = F(K)$, converges to K^* asymptotically, and with a quadratic convergence rate (asymptotically as $K \to K^*$). Note that for our one-dimensional problem, the region of stability is the interval (K_S, ∞) that is characterized by the single point K_S where F has derivative equal to 1; see Fig. 1.5.9.

For multidimensional problems, the region of stability may not be characterized as easily. Still, however, it is generally true that *the region of stability is enlarged as the length of the lookahead increases*.

Indeed, with increased lookahead, the effective starting point

$$F^{\ell-1}(\tilde{K})$$

is pushed more and more within the region of stability. In particular, *for any given $K \geq 0$, the corresponding ℓ-step lookahead policy will be stable for all ℓ larger than some threshold*; see Fig. 1.5.8. The book [Ber22a], Section 3.3, contains a broader discussion of the region of stability and the role of multistep lookahead in enhancing it; see also Exercise 1.8.

Newton Step Interpretation of Approximation in Value Space in General Infinite Horizon Problems

The interpretation of approximation in value space as a Newton step, and related notions of stability that we have discussed in this section admit a broad generalization to the infinite horizon problems that we consider in this book and beyond. The key fact in this respect is that our DP problem formulation allows arbitrary state and control spaces, both discrete and continuous, and can be extended even further to general abstract models with a DP structure; see the abstract DP book [Ber22b].

Within this context, the Riccati operator is replaced by an abstract Bellman operator, and valuable insight can be obtained from graphical interpretations of the Bellman equation, the VI and PI algorithms, one-step and multistep approximation in value space, the region of stability, and exceptional behavior; see the book [Ber22a] for an extensive discussion. Naturally, the graphical interpretations and visualizations are limited to one dimension. However, they provide insight, and motivate conjectures and mathematical analysis, much of which is given in the book [Ber20a].

The Importance of the First Step in Multistep Lookahead

The Newton step interpretation of approximation in value space leads to an important insight into the special character of the initial step in ℓ-step lookahead implementations. In particular, *it is only the first step that acts as the Newton step*, and needs to be implemented with precision; cf. Fig. 1.5.5. The subsequent $\ell-1$ steps are a sequence of value iterations starting with $\tilde{J}$, and only serve to enhance the quality of the starting point of the Newton step. As a result, *their precise implementation is not critical*, a major point in the narrative of the author's book [Ber22a].

This idea suggests that we can simplify (within reason) the lookahead steps after the first with small (if any) performance loss for the multistep lookahead policy. An important example of such a simplification is the use of certainty equivalence, which will be discussed later in various contexts (Sections 1.6.7, 2.7.2, 2.8.3). Other possibilities include the "pruning" of the lookahead tree after the first step; see Section 2.4. In practical terms, simplifications after the first step of the multistep lookahead can save a lot of on-line computation, which can be fruitfully invested in extending the length of the lookahead. This insight is supported by substantial computational experimentation, starting with the paper by Bertsekas and Castañon [BeC98], which verified the beneficial effect of using certainty equivalence after the first step.

1.5.2 Rollout and Policy Iteration

We will now consider the rollout algorithm for the linear quadratic problem, starting from a linear stable base policy μ. It generates the rollout policy

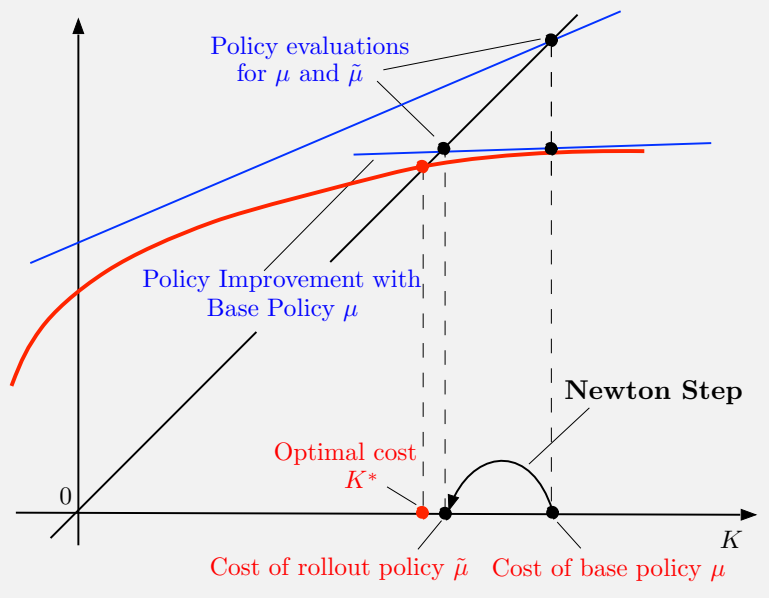

Figure 1.5.10 Illustration of the rollout algorithm for the linear quadratic problem. Starting from a linear stable base policy μ, it generates a stable rollout policy $\tilde{\mu}$. The quadratic cost coefficient of $\tilde{\mu}$ is obtained from the quadratic cost coefficient of μ with a Newton step for solving the Riccati equation.

$\tilde{\mu}$ by using a policy improvement operation, which by definition, yields the one-step lookahead policy that corresponds to terminal cost approximation $\tilde{J} = J_\mu$. Figure 1.5.10 illustrates the rollout algorithm. It can be seen from the figure that the rollout policy is in fact an improved policy, in the sense that $J_{\tilde{\mu}}(x) \leq J_\mu(x)$ for all x. Among others, this implies that the rollout policy is stable, since μ is assumed stable so that $J_\mu(x) < \infty$ for all x.

Since the rollout policy is a one-step lookahead policy, it can also be described using the formulas that we developed earlier in this section. In particular, let the base policy have the form

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

where L_0 is a scalar. We require that the base policy must be stable, i.e., $|a + bL_0| < 1$. From our earlier calculations, we have that the cost function of μ^0 is

$$J_{\mu^0}(x) = K_0 x^2, \tag{1.54}$$

where

$$K_0 = \frac{q + rL_0^2}{1 - (a + bL_0)^2}. \tag{1.55}$$

Moreover, the rollout policy μ^1 has the form $\mu^1(x) = L_1 x$, where

$$L_1 = -\frac{abK_0}{r + b^2 K_0}; \tag{1.56}$$

cf. Eqs. (1.47)-(1.48).

The PI algorithm is simply the repeated application of nontruncated rollout, and generates a sequence of stable linear policies $\{\mu^k\}$. By replicating our earlier calculations, we see that the policies have the form

$$\mu^k(x) = L_k x, \qquad k = 0, 1, \ldots,$$

where L_k is generated by the iteration

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

with K_k given by

$$K_k = \frac{q + rL_k^2}{1 - (a + bL_k)^2},$$

[cf. Eqs. (1.55)-(1.56)].

The corresponding cost function sequence is

$$J_{\mu^k}(x) = K_k x^2;$$

cf. Eq. (1.54). Part of the classical linear quadratic theory is that J_{μ^k} converges 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, assuming that the initial policy is linear and stable. The convergence rate of the sequence $\{K_k\}$ is quadratic, as indicated earlier. This result was proved by Kleinman [Kle68] for the continuous-time multidimensional version of the linear quadratic problem, and it was extended later to more general problems; see the references given in the books [Ber20a] and [Ber22a] (Kleinman gives credit to Bellman and Kalaba [BeK65] for the one-dimensional version of his results).

Truncated Rollout

An m-step truncated rollout scheme with a stable linear base policy $\mu(x) = Lx$, one-step lookahead minimization, and terminal cost approximation $\tilde{J}(x) = \tilde{K}x^2$ is geometrically interpreted as in Fig. 1.5.11. The truncated rollout policy $\tilde{\mu}$ is obtained by starting at $\tilde{K}$, executing m VI steps using μ, followed by a Newton step for solving the Riccati equation.

We mentioned some interesting performance issues in our discussion of truncated rollout in Section 1.1. In particular we noted that:

(a) Lookahead by rollout may be an economic substitute for lookahead by minimization: it may achieve a similar performance for the truncated rollout policy at a much reduced computational cost.

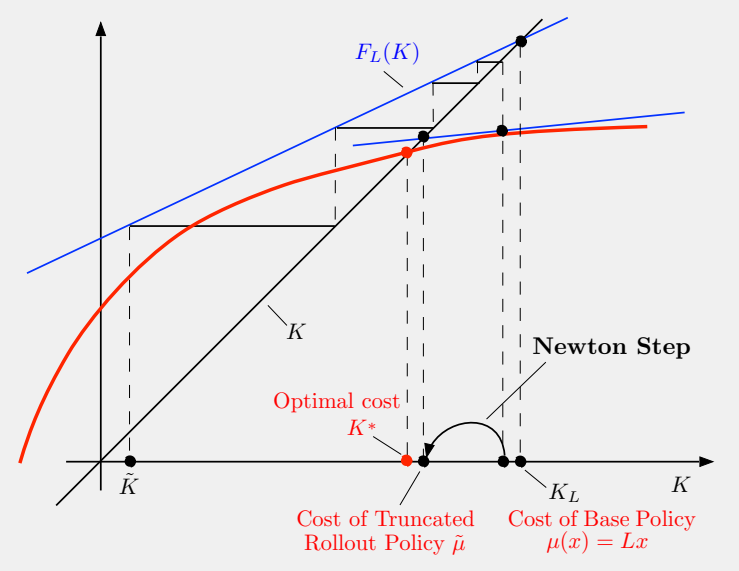

Figure 1.5.11 Illustration of truncated rollout with one-step lookahead minimization, a stable base policy $\mu(x) = Lx$, and terminal cost approximation $\tilde{K}$ for the linear quadratic problem. In this figure the number of rollout steps is $m = 4$.

(b) Lookahead by rollout with a stable policy has a beneficial effect on the stability properties of the lookahead policy.

These statements are difficult to establish analytically in some generality. However, they can be intuitively understood in the context with our one-dimensional linear quadratic problem, using geometrical constructions like the one of Fig. 1.5.11. They are also consistent with the results of computational experimentation. We refer to the monograph [Ber22a] for further discussion.

Double Newton Step - Rollout on Top of Approximation in Value Space

Given a cost function approximation $\tilde{K}$ that defines the policy $\tilde{\mu}(x) = L_{\tilde{K}}x$, it is possible to consider rollout that uses $\tilde{\mu}$ as a base policy. This can be viewed as rollout built on top of approximation in value space, and is referred to as a *double Newton step*; it is a Newton step that starts from the result of the Newton step that starts from $\tilde{K}$ (see Fig. 1.5.12). The double Newton step is much more powerful than approximation in value space with two-step lookahead starting from $\tilde{K}$, which amounts to a value iteration followed by a Newton step. Moreover, the idea of a double Newton step extends to general infinite horizon problems.

Note that it is also possible to consider variants of rollout on top of approximation in value space, such as truncated, simplified, and multi-

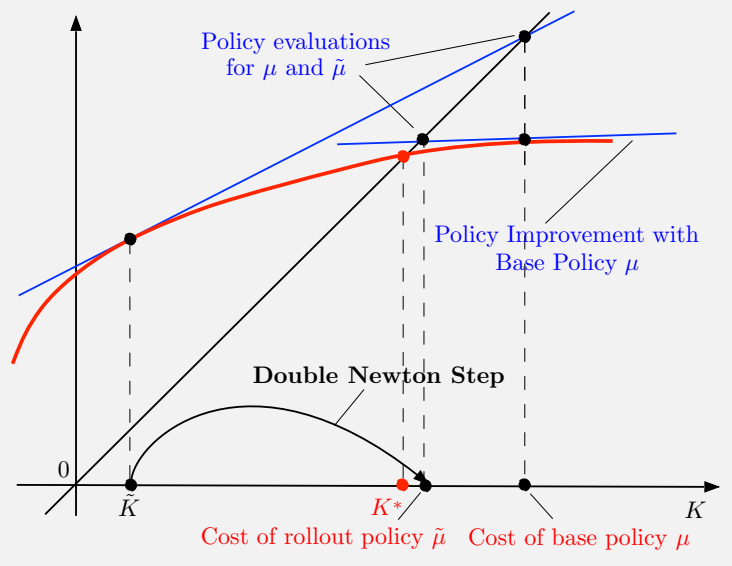

Figure 1.5.12 Illustration of a double Newton step. Here, a base policy μ is obtained by one-step lookahead using cost function approximation $\tilde{J}(x) = \tilde{K}x^2$. The cost function of the corresponding rollout policy $\tilde{\mu}$ is obtained with two successive Newton steps starting from $\tilde{K}$.

step lookahead versions. An important example of the truncated version is the 1996 TD-Gammon architecture [TeG96], where the base policy is obtained through approximation in value space with a terminal cost function approximation that is constructed off-line using a neural network.

1.5.3 Local and Global Error Bounds for Approximation in Value Space

In approximation in value space, an important analytical issue is to quantify the level of suboptimality of the one-step or multistep lookahead policy obtained. It is thus important to understand the character of the critical mapping between the approximation error $\tilde{J} - J^*$ and the performance error $J_{\tilde{\mu}} - J^*$, where as earlier, $J_{\tilde{\mu}}$ is the cost function of the lookahead policy $\tilde{\mu}$ and J^* is the optimal cost function.

There is a classical one-step lookahead error bound for the case of an α-discounted problem with finite state space X, which has the form

$$\|J_{\tilde{\mu}} - J^*\| \leq \frac{2\alpha}{1-\alpha}\|\tilde{J} - J^*\|; \tag{1.57}$$

where $\|\cdot\|$ denotes the maximum norm,

$$\|J_{\tilde{\mu}} - J^*\| = \max_{x \in X} \big|J_{\tilde{\mu}}(x) - J^*(x)\big|, \qquad \|\tilde{J} - J^*\| = \max_{x \in X} \big|\tilde{J}(x) - J^*(x)\big|;$$

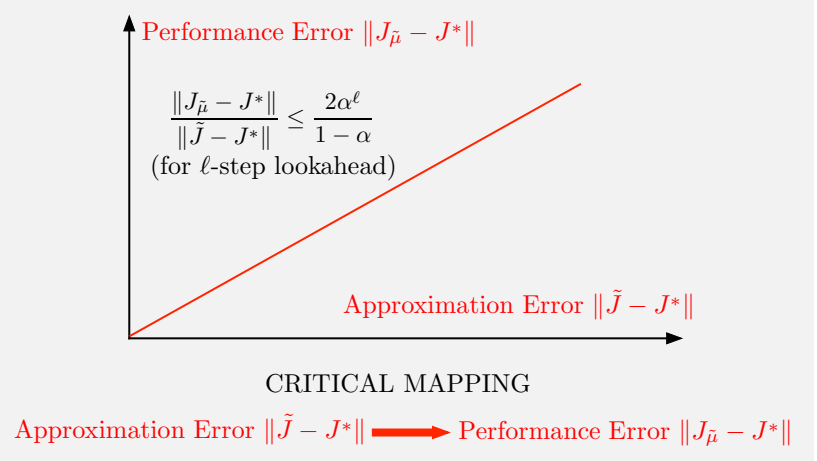

Figure 1.5.13 Illustration of the linear error bound (1.58) for ℓ-step lookahead approximation in value space. For $\ell = 1$, we obtain the one-step bound (1.57).

see e.g., [Ber19a], Prop. 5.1.1. The bound (1.57) predicts a linear relation between the size of the approximation error $\|\tilde{J} - J^*\|$ and the performance error $\|J_{\tilde{\mu}} - J^*\|$. For a generalization, we may view ℓ-step lookahead as one-step lookahead with a terminal cost function $T^{\ell-1}\tilde{J}$, i.e., $\tilde{J}$ transformed by $\ell - 1$ value iterations. We then obtain the ℓ-step bound

$$\|J_{\tilde{\mu}} - J^*\| \leq \frac{2\alpha^\ell}{1-\alpha}\|\tilde{J} - J^*\|. \tag{1.58}$$

The linear bounds (1.57)-(1.58) are illustrated in Fig. 1.5.13, and apply beyond the α-discounted case, to problems where the Bellman equation involves a contraction mapping over a subset of functions; see the RL book [Ber19a], Section 5.9.1, or the abstract DP book [Ber22b], Section 2.2.

Unfortunately, *the linear error bounds are very conservative, and do not reflect practical reality, even qualitatively so.* The main reason is that they are *global* error bounds, i.e., they hold for all $\tilde{J}$, even the worst possible. In practice, $\tilde{J}$ is often chosen sufficiently close to J^*, so that the error $J_{\tilde{\mu}} - J^*$ behaves consistently with the superlinear convergence rate of the Newton step that starts at $\tilde{J}$. In other words, for $\tilde{J}$ relatively close to J^*, we have the *local* estimate

$$\|J_{\tilde{\mu}} - J^*\| = o\big(\|\tilde{J} - J^*\|\big), \tag{1.59}$$

illustrated in Fig. 1.5.14.

A salient characteristic of this superlinear relation is that the performance error rises rapidly outside the region of superlinear convergence of Newton's method. Note that small improvements in the quality of $\tilde{J}$ (e.g., better sampling methods, improved confidence intervals, and the like, without changing the approximation architecture) have *little effect, both inside and outside the region of convergence*.

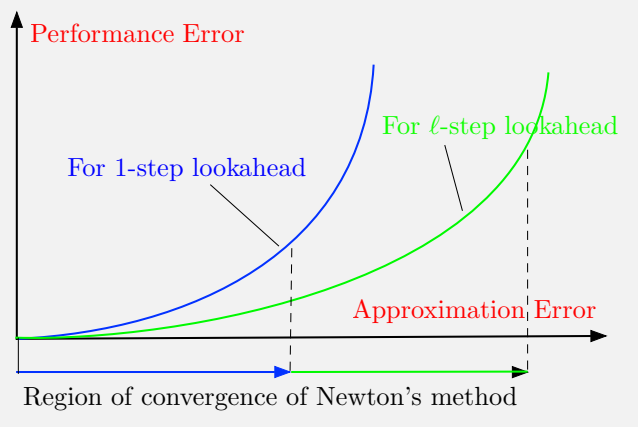

Figure 1.5.14 Schematic illustration of the correct superlinear error bound (1.59) for the case of ℓ-step lookahead approximation in value space scheme. The performance error rises rapidly outside the region of convergence of Newton's method [the illustration in the figure is not realistic; in fact the region of convergence is not bounded as it contains lines of the form γe, where γ is a scalar and e is the unit vector (all components equal to 1)]. Note that this region expands as the size of lookahead ℓ increases. Furthermore, with long enough lookahead ℓ, the ℓ-step lookahead policy $\tilde{\mu}$ can be shown to be *exactly optimal* for many problems of interest; this is a theoretical result, which holds for α-discounted finite-state problems, among others, and has been known since the 60s-70s (Prop. 2.3.1 of [Ber22a] proves a general form of this result that applies beyond discounted problems).

In practical terms, *there is often a huge difference, both quantitative and qualitative, between the linear error bounds (1.57)-(1.58) and the superlinear error bound (1.59)*. Moreover, the linear bounds, despite their popularity in academia, often misdirect academic research and confuse practitioners.† Note that as we have mentioned earlier, the qualitative performance behavior predicted in Fig. 1.5.14 holds very broadly in ap-

† A study by Laidlaw, Russell, and Dragan [LRD23] has assessed the practical performance of popular methods on a set of 155 problems, and found wide disparities relative to theoretical predictions. Quoting from this paper: "we find that prior bounds do not correlate well with when deep RL succeeds vs. fails."

The study goes on to assert the importance of long multistep lookahead (the size of ℓ) in stabilizing the performance of approximation in value space schemes. It also confirms computationally a known theoretical result, namely that with long enough lookahead ℓ, the ℓ-step lookahead policy $\tilde{\mu}$ is exactly optimal (but the required length of ℓ depends on the approximation error $\tilde{J} - J^*$). This fact has been known since the 60s-70s for α-discounted finite-state problems. A generalization of this result is given as Prop. 2.3.1 of the abstract DP book [Ber22b]; see also Section A.4 of the book [Ber22a], which discusses the convergence of Newton's method for systems of equations that involve nondifferentiable mappings (such as the Bellman operator).

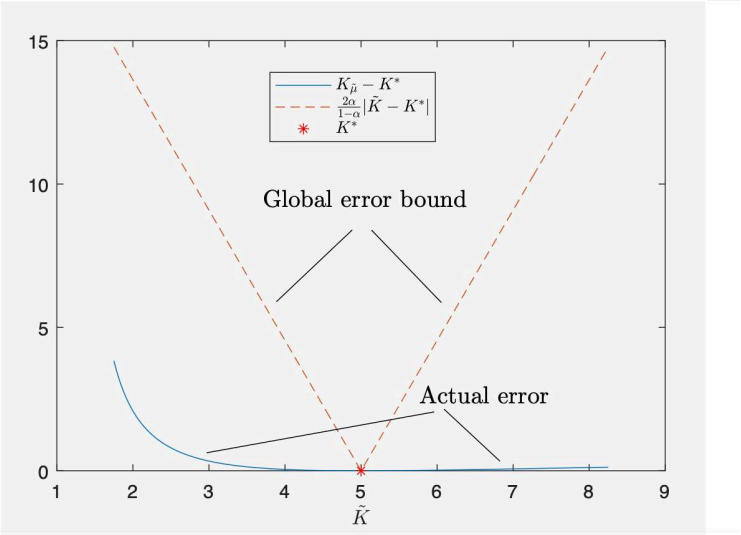

Figure 1.5.15 Illustration of the global error bound for the one-step lookahead error $K_{\tilde{\mu}} - K^*$ as a function of $\tilde{K}$, compared with the true error obtained by one step of Newton's method starting from $\tilde{K}$; cf. Example 1.5.1.

The problem data are $a = 2$, $b = 2$, $q = 1$, and $r = 5$. With these numerical values, we have $K^* = 5$ and the region of stability is (S, ∞) with $S = 1.25$. The modulus of contraction α used in the figure is computed at $\overline{S} = S + 0.5$. Depending on the chosen value of $\overline{S}$, α can be arbitrarily close to 1, but decreases as $\overline{S}$ increases. Note that the error $K_{\tilde{\mu}} - K^*$ is much smaller when $\tilde{K}$ is larger than K^* than when it is lower, because the slope of F diminishes as K increases. This is not reflected by the global error bound.

proximation in value space, because it relies on notions of abstract DP that apply very generally, for arbitrary state spaces, control spaces, and other problem characteristics; see the abstract DP book [Ber22a].

We illustrate the failure of the linear error bound (1.57) to predict the performance of the one-step lookahead policy with an example given in Fig. 1.5.15.

Example 1.5.1 (Global and Actual Error Bounds for a Linear Quadratic Problem)

Consider the one-dimensional linear quadratic problem, involving the system $x_{k+1} = ax_k + bu_k$, and the cost per stage $qx_k^2 + ru_k^2$. We will consider one-step lookahead, and a quadratic cost function approximation

$$\tilde{J}(x) = \tilde{K}x^2,$$

with $\tilde{K}$ within the region of stability, which is some interval of the form (S, ∞). The Riccati operator is

$$F(K) = \frac{a^2 rK}{r + b^2 K} + q,$$

and the one-step lookahead policy $\tilde{\mu}$ has cost function

$$J_{\tilde{\mu}}(x) = K_{\tilde{\mu}} x^2,$$

where $K_{\tilde{\mu}}$ is obtained by applying one step of Newton's method for solving the Riccati equation $K = F(K)$, starting at $K = \tilde{K}$.

Let S be the boundary of the region of stability, i.e., the value of K at which the derivative of F with respect to K is equal to 1:

$$\frac{\partial F(K)}{\partial K}\Big|_{K=S} = 1.$$

Then the Riccati operator F is a contraction within any interval $[\overline{S}, \infty)$ with $\overline{S} > S$, with a contraction modulus α that depends on $\overline{S}$. In particular, α is given by

$$\alpha = \frac{\partial F(K)}{\partial K}\Big|_{K=\overline{S}}$$

and satisfies $0 < \alpha < 1$ because $\overline{S} > S$, and the derivative of F is positive and monotonically decreasing to 0 as K increases to ∞.

The error bound (1.57) can be rederived for the case of quadratic functions and can be rewritten in terms of quadratic cost coefficients as

$$K_{\tilde{\mu}} - K^* \leq \frac{2\alpha}{1-\alpha}|\tilde{K} - K^*|, \tag{1.60}$$

where $K_{\tilde{\mu}}$ is the quadratic cost coefficient of the lookahead policy $\tilde{\mu}$ [and also the result of a Newton step for solving the fixed point Riccati equation $F = F(K)$ starting from $\tilde{K}$]. A plot of $(K_{\tilde{\mu}} - K^*)$ as a function of $\tilde{K}$, compared with the bound on the right side of this equation is shown in Fig. 1.5.15. It can be seen that $(K_{\tilde{\mu}} - K^*)$ exhibits the qualitative behavior of Newton's method, which is very different than the bound (1.60). An interesting fact is that the bound (1.60) depends on α, which in turn depends on how close $\tilde{K}$ is to the boundary S of the region of stability, while the local behavior of Newton's method is independent of S.

How Approximation in Value Space Can Fail and What to Do About It

Let us finally discuss the most common way that approximation in value space can fail. Consider the case where the terminal cost approximation $\tilde{J}$ is obtained through training with data of an approximation architecture such as a neural network (e.g., as in AlphaZero and TD-Gammon). Then there are three components that determine the approximation error $\tilde{J} - J^*$:

(a) The *power of the architecture*, which roughly speaking is a measure of the error that would be obtained if infinite data were available and were used optimally to obtain $\tilde{J}$.

(b) The *error degradation due the limited availability of training data.*

(c) The additional *error degradation due to imperfections in the training methodology*.

Thus *if the architecture is not powerful enough to bring $\tilde{J} - J^*$ within the region of convergence of Newton's method, approximation in value space with one-step lookahead will likely fail, no matter how much data is collected and how effective the associated training method is*.

In this case, there are two potential practical remedies:

(1) Use a more powerful architecture/neural network for representing $\tilde{J}$.

(2) Extend the combined length of the lookahead minimization and truncated rollout in order to bring the effective value of $\tilde{J}$ within the region of convergence of Newton's method.

The first remedy typically requires a deep neural network or transformer, which uses more weights and requires more expensive training (see Chapter 3).† The second remedy requires longer on-line computation and/or simulation, which may run up against some practical real-time implementation constraint. Parallel computation and sophisticated multistep lookahead implementation methods may help to mitigate these requirements (see Chapter 2).

1.6 EXAMPLES, REFORMULATIONS, 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], [Ber20a], as well as in the neuro-dynamic programming book [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.

1.6.1 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.

† For a recent example of implementation of a grandmaster-level chess program with *one-step lookahead* and a huge-size (270M parameter) neural network position evaluator, see Ruoss et al. [RDM24].

Generally, a practical problem is a prime candidate for a DP formulation if it involves multiple sequential decisions, which are separated by feedback, i.e., by observations that can be used to 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 (see, e.g., the traveling salesman Example 1.2.3). 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 Chapter 2. We will argue there that rollout and its variations, when suitably applied, can improve substantially on 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 not be considered 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 generally relates to the approximation in policy space approach, which we have discussed in Section 1.3.3 and we will discuss further in Section 3.4; see also Section 5.7 of the RL book [Ber19a].

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.

A Few Words about the Choice of an RL Method

An attractive aspect of the current RL methodology, inherited by the generality of our DP formulation, is that it can address a very broad range of challenging problems, deterministic as well as stochastic, discrete as well as continuous, etc. However, in the practical application of RL methods one has to contend with limited theoretical guarantees. In particular, several of the RL methods that have been successful in practice have less than solid performance properties, and may not work on a given problem, even one of the type for which they are designed.

This is a reflection of the state of the art in the field: *there are no methods that are guaranteed to work for all or even most DP problems*. However, there are enough methods to try on a given problem with a reasonable chance of success in the end (after some heuristic and problem specific tuning). For this reason, it is important to develop insight into the inner workings of various methods, as a means of selecting the proper type of methodology to try on a given problem.†

A related consideration is the context within which a method is applied. In particular, is it a single problem that is being addressed, such as chess that has fixed rules and a fixed initial condition, or is it a family of related problems that must be periodically be solved with small variations in its data or its initial conditions? Also, are the problem data fixed or may they change over time as the system is being controlled?

Generally, convenient but relatively unreliable methods, which can be tuned to the problem at hand, may be tried with a reasonable chance of success if a single problem is addressed. Similarly, RL methods that require extensive tuning of parameters, including ones that involve approximation in policy space and the use of neural networks, may be well suited for a stable problem environment and a single problem solution. However, they not well suited for problems with a variable environment and/or real-time changes of model parameters. For such problems, RL methods based on approximation in value space and on-line play, possibly involving on-line replanning, are much better suited.

Note also that even when on-line replanning is not needed, on-line play may improve substantially the performance of off-line trained policies, so we may wish to use it in conjunction with off-line training. This is due to the Newton step that is implicit in one-step or multistep lookahead minimization, cf. our discussion of the AlphaZero and TD-Gammon architectures in Section 1.1. Of course the computational requirements of an

† Aside from insight and intuition, it is also important to have a foundational understanding of the analytical principles of the field and of the mechanisms underlying the central computational methods. The role of the theory in this respect is to structure mathematically the methodology, guide the art, and delineate the sound from the flawed ideas.

on-line play method may be substantial and have to be taken into account when assessing its suitability for a particular application. In this connection, deterministic problems are better suited than stochastic problems for on-line play. Moreover, methods that are well-suited for parallel computation, and/or involve the use of certainty equivalence approximations are generally better suited for a stochastic control environment.

1.6.2 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) and have played an important role in the development of the RL methodology.†

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.

Example 1.6.1 (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

† 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 problem.

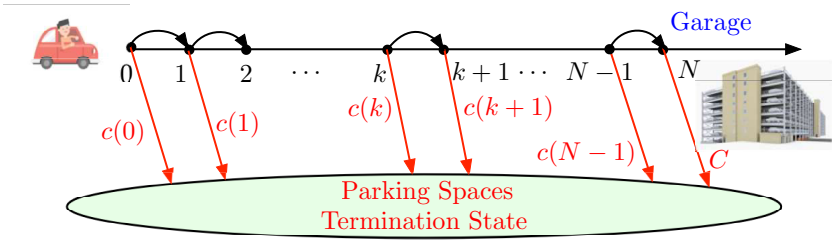

Figure 1.6.1 Cost structure of the parking problem. The driver may park at space $k = 0, 1, \ldots, N-1$ at cost $c(k)$, if the space is free, or continue to the next space $k+1$ at no cost. At space N (the garage) the driver must park at cost C.

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. 1.6.1. At each stage $k = 1, \ldots, N-1$, we have three states: the artificial termination state t, 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 [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 $c(k) \leq \hat{J}_{k+1}$.

This is an example of DP simplification that occurs when the state involves components that are not affected by the choice of control, and will be addressed in the next section.

1.6.3 General Discrete Optimization Problems

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, similar to the preceding four queens example, the scheduling Example 1.2.1, and the traveling salesman Examples 1.2.2 and 1.2.3. This formulation often leads to an intractable exact DP computation because of an exponential explosion of the number of states as time progresses. However, a reformulation to a discrete optimal control problem 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.

Let us now extend the ideas of the examples just noted to the general discrete optimization problem:

$$\begin{aligned} &\text{minimize} \quad G(u) \\ &\text{subject to} \quad u \in U, \end{aligned} \tag{1.61}$$

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

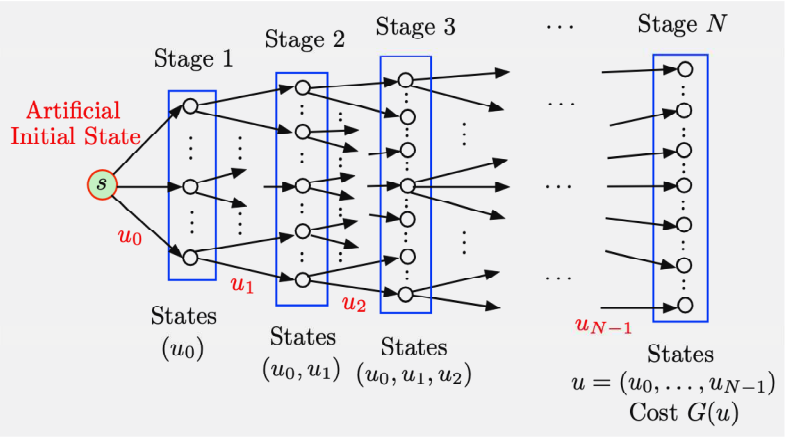

Figure 1.6.2 Formulation of a discrete optimization problem as a DP problem with N stages. There is a cost $G(u)$ only at the terminal stage on the arc connecting an N-solution $u = (u_0, \ldots, u_{N-1})$ upon reaching the terminal state. Note that there is only one incoming arc at each node.

solution is called a *k-solution*. We associate k-solutions with the kth stage of the finite horizon discrete optimal control problem shown in Fig. 1.6.2. 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

$$U_0 = \big\{\tilde{u}_0 \mid \text{there exists a solution of the form } (\tilde{u}_0, \tilde{u}_1, \ldots, \tilde{u}_{N-1}) \in U\big\}. \tag{1.62}$$

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

$$U_k(u_0, \ldots, u_{k-1}) = \big\{u_k \mid \text{for some } \overline{u}_{k+1}, \ldots, \overline{u}_{N-1} \text{ we have } (u_0, \ldots, u_{k-1}, u_k, \overline{u}_{k+1}, \ldots, \overline{u}_{N-1}) \in U\big\}. \tag{1.63}$$

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 [we do not exclude the possibility that the set (1.63) is empty]. The last stage corresponds to the N-solutions $u = (u_0, \ldots, u_{N-1})$, and the terminal cost is $G(u)$; see Fig. 1.6.2. 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

$$u_k^* \in \arg\min_{u_k \in U_k(u_0^*,\ldots,u_{k-1}^*)} J_{k+1}^*(u_0^*,\ldots,u_{k-1}^*,u_k), \quad k = 0,\ldots,N-1, \tag{1.64}$$

where U_0 is given by Eq. (1.62), and U_k is given by Eq. (1.63): first compute u_0^*, then u_1^*, and so on up to u_{N-1}^*; cf. Eq. (1.8).

Of course here the number of states typically grows exponentially with N, but we can use the DP minimization (1.64) as a starting point for 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. (1.64). 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 turns out to be a very simple and effective approach for approximate combinatorial optimization.

Let us finally note that while we have used a general cost function G and constraint set U in our discrete optimization model of this section, in many problems G and/or U may have a special (e.g., additive) structure, which is consistent with a sequential decision making process and may be computationally exploited. The traveling salesman Example 1.2.2 is a case in point, where G consists of the sum of N components (the intercity travel costs), one per stage. Our next example deals with a problem of great current interest.

Example 1.6.2 (A Large Language Model Based on N-Grams)

Let us consider an N-gram model, whereby a text string consisting of N words is transformed into another string of N words by adding a word at the front of the string and deleting the word at the back of the string. We view the text strings as states of a dynamic system affected by the added word choice, which we view as the control. We denote by x_k the string obtained at

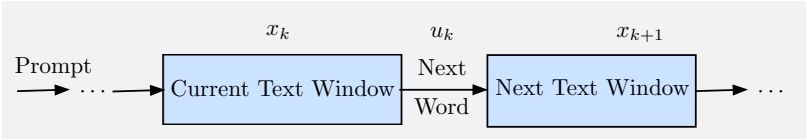

Figure 1.6.3 Schematic visualization of an LLM problem based on N-grams.

time k, and by u_k the word added at time k. We assume that u_k is chosen from a given set $U(x_k)$. Thus we have a controlled dynamic system, which is deterministic and is described by an equation of the form

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

where f specifies the operation of adding u_k at the front of x_k and removing the word at the back of x_k. The initial string x_0 is assumed given.

If we have a cost function G by which to evaluate a text string, we can pose a DP problem with either a finite or an infinite horizon. For example if the string evolution terminates after exactly N steps, we obtain the finite horizon problem of minimizing the function $G(x_N)$ of the final text string x_N. In this case, x_N is obtained after we have a chance to change successively all the words of the initial string x_0, subject to the constraints $u_k \in U(x_k)$.

Another possibility is to introduce a termination action, whereby addition/deletion of words is optionally stopped at some time and the final text string x is obtained with cost $G(x)$. In such a problem formulation, we may also include an additive stage cost that depends on u. This is an infinite horizon formulation that involves an additional termination state t in the manner of Section 1.6.2.

Note that in both the finite and the infinite horizon formulation of the problem, the initial string x_0 may include a "prompt," which may be subject to optimization through some kind of "prompt engineering." Depending on the context, this may include the use of another optimization or heuristic algorithm, perhaps unrelated to DP, which searches for a favorable prompt from within a given set of choices.

Interesting policies for the preceding problem formulation may be provided by a neural network, such as a Generative Pretrained Transformer (GPT). In our terms, the GPT can be viewed simply as a policy that generates next words. This policy may be either deterministic, i.e., $u_k = \mu(x_k)$ for some function μ, or it may be a "randomized" policy, which generates u_k according to a probability distribution that depends on x_k. Our DP formulation can also form the basis for policy improvement algorithms such as rollout, which aim to improve the quality of the output generated by the GPT. Another, more ambitious, possibility is to consider an approximate, neural network-based, policy iteration/self-training scheme, such as the ones discussed earlier, based on the AlphaZero/TD-Gammon architecture. Such a scheme generates a sequence of GPTs, with each GPT trained with data provided by the preceding GPT, a form of self-learning in the spirit of the AlphaZero and TD-Gammon policy iteration algorithms, cf. Section 1.1.

It is also possible to provide an infinite horizon formulation of the general discrete optimization problem

$$\begin{aligned} &\text{minimize} \quad G(u) \\ &\text{subject to} \quad u \in U, \end{aligned} \tag{1.65}$$

where U is a finite set of feasible solutions, $G(u)$ is a cost function, and u consists of N components, $u = (u_0, \ldots, u_{N-1})$; cf. Eq. (1.61). To this end, we introduce a termination state t that the system enters after N steps. At step k, the component u_k is selected subject to $u_k \in U_k(u_0, \ldots, u_{k-1}$, where the constraint set $U_k(u_0, \ldots, u_{k-1}$ is given by Eq. (1.63). This is a special case of a general finite to infinite horizon stochastic DP problem reformulation, which we describe in the next section.

1.6.4 General Finite to Infinite Horizon Reformulation

There is a conceptually important reformulation that transforms a finite horizon problem, possibly involving a nonstationary system and cost per stage, to an equivalent infinite horizon problem. It is based on introducing an expanded state space, which is the union of the state spaces of the finite horizon problem plus an artificial cost-free termination state that the system moves into at the end of the horizon. This reformulation is of great conceptual value, as it provides a mechanism to bring to bear ideas that can be most conveniently understood within an infinite horizon context. For example, it helps to understand the synergy of off-line training and on-line play based on Newton's method, and the related insights that explain the good performance of rollout algorithms in practice.

To define the reformulation, let us consider the N-stage horizon stochastic problem of Section 1.3.1, whose system has the form

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, \ldots, N-1, \tag{1.66}$$

and let us denote by X_k, $k = 0, \ldots, N$, and U_k, $k = 0, \ldots, N-1$, the corresponding state spaces and control spaces, respectively. We introduce an artificial termination state t, and we consider an infinite horizon problem with state and control spaces X and U given by

$$X = \left(\cup_{k=0}^{N} X_k\right) \cup \{t\}, \qquad U = \cup_{k=0}^{N-1} U_k; \tag{1.67}$$

see Fig. 1.6.4.

The system equation and the control constraints of this problem are also reformulated so that states in X_k, $k = 0, \ldots, N-1$, are mapped to states in X_{k+1}, according to Eq. (1.66), while states $x_N \in X_N$ are mapped to the termination state t at cost $g_N(x_N)$. Upon reaching t, the state stays at t at no cost. Thus the policies of the infinite horizon problem map states

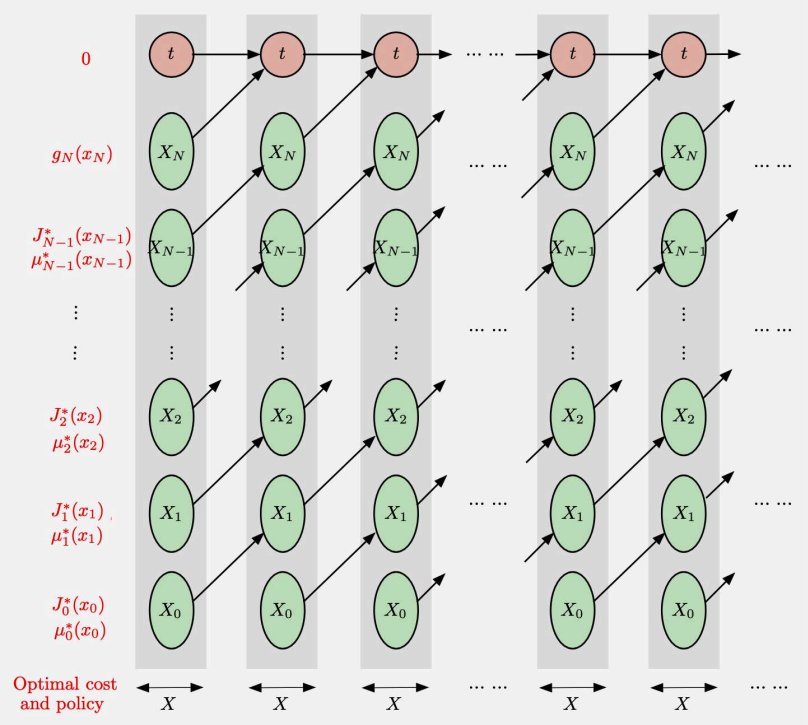

Figure 1.6.4 Illustration of the infinite horizon equivalent of a finite horizon problem. The state space is $X = \left(\cup_{k=0}^{N} X_k\right) \cup \{t\}$, and the control space is $U = \cup_{k=0}^{N-1} U_k$. Transitions from states $x_k \in X_k$ lead to states in $x_{k+1} \in X_{k+1}$ according to the system equation $x_{k+1} = f_k(x_k, u_k, w_k)$, and they are stochastic when they involve the random disturbance w_k. The transition from states $x_N \in X_N$ lead deterministically to the termination state at cost $g_N(x_N)$. The termination state t is cost-free and absorbing.

The infinite horizon optimal cost $J^*(x_k)$ and optimal policy $\mu^*(x_k)$ at state $x_k \in X_k$ of the infinite horizon problem are equal to optimal cost-to-go $J^*_k(x_k)$ and optimal policy $\mu^*_k(x_k)$ of the finite horizon problem.

$x_k \in X_k$ to controls in $U_k(x_k) \subset U_k$, and consist of functions $\mu_k(x_k)$ that are policies of the finite horizon problem. Moreover, the Bellman equation for the infinite horizon problem is identical to the DP algorithm for the finite horizon problem.

It can be seen that the optimal cost and optimal control, $J^*(x_k)$ and $\mu^*(x_k)$, at a state $x_k \in X_k$ in the infinite horizon problem are equal to the optimal cost-to-go $J^*_k(x_k)$ and optimal control $\mu^*_k(x_k)$ of the original finite horizon problem, respectively; cf. Fig. 1.6.4. Moreover approximation in value space and rollout in the finite horizon problem translate to infinite horizon counterparts, and can be understood as Newton steps for solving the Bellman equation of the infinite horizon problem (or equivalently the DP algorithm of the finite horizon problem).

In summary, finite horizon problems can be viewed as infinite horizon problems with a special structure that involves a termination state t, and the state and control spaces of Eq. (1.67), as illustrated in Fig. 1.6.4. The Bellman equation of the infinite horizon problem coincides with the DP algorithm of the finite horizon problem. The PI algorithm for the infinite horizon problem can be translated directly to a PI algorithm for the finite horizon problem, involving repeated policy evaluations and policy improvements. Finally, the Newton step interpretations for approximation in value space and rollout schemes for the infinite horizon problem have straightforward analogs for finite horizon problems, and explain the powerful cost improvement mechanism that underlies the rollout algorithm and its variations.

1.6.5 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 formulation 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 intuitive 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

$$x_{k+1} = f_k(x_k, x_{k-1}, u_k, u_{k-1}, w_k), \qquad k = 1, \ldots, N-1, \tag{1.68}$$

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

If we introduce additional state variables y_k and s_k, and we make the identifications $y_k = x_{k-1}$, $z_k = u_{k-1}$, the system equation (1.68) yields

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, y_k, u_k, z_k, w_k) \\ x_k \\ u_k \end{pmatrix}. \tag{1.69}$$

By defining $\tilde{x}_k = (x_k, y_k, z_k)$ 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. (1.69).

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 is

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

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

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

$$J_k^*(x_k, y_k) = \min_{u_k \in U_k(x_k)} E_{w_k}\Bigg\{ g_k(x_k, u_k, w_k) + \sum_{i=1}^{m} p_i J_{k+1}^*\big(f_k(x_k, u_k, w_k), i\big) \mid y_k \Bigg\}, \tag{1.70}$$

where y_k may take the values $1, \ldots, m$, and the expectation 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" (see also the parking Example 1.6.1). 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 parking Example 1.6.1, a DP algorithm that generates $\hat{J}_k(x_k)$ can be obtained, and has the following form:

$$\hat{J}_k(x_k) = E_{y_k}\Bigg\{ \min_{u_k \in U_k(x_k, y_k)} E_{w_k}\Big\{g_k(x_k, y_k, u_k, w_k) + \hat{J}_{k+1}\big(f_k(x_k, y_k, u_k, w_k)\big) \;\Big|\; x_k, y_k, u_k\Big\}\Big|\; x_k\Bigg\}. \tag{1.71}$$

This is a consequence of the calculation

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

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 (1.71) 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. (1.71). 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. (1.70),

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

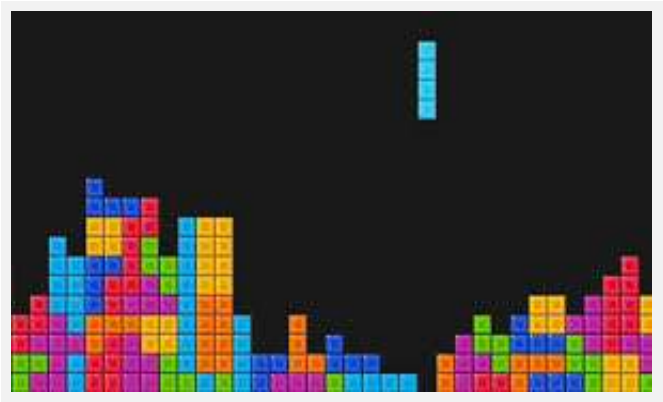

Figure 1.6.5 Illustration of a tetris board.

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. (1.70).

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. (1.71). Here is an example of this type.

Example 1.6.3 (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. 1.6.5). 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 (1.71) is executed over the space of board positions x and has the intuitive form

$$\hat{J}_k(x) = \sum_y p(y) \max_u \Big[g(x,y,u) + \hat{J}_{k+1}\big(f(x,y,u)\big) \Big], \qquad \text{for all } x, \qquad (1.72)$$

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 (1.72) 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 (1.72), 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.†

1.6.6 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 of x_k. However, in many practical settings, this assumption is unrealistic

† Tetris is generally considered to be an interesting and challenging stochastic testbed for RL algorithms, and has received a lot of attention over a period spanning 20 years (1995-2015), starting with the papers [TsV96], [BeI96], and the neuro-dynamic programming book [BeT96], and ending with the papers [GGS13], [SGG15], which contain many references to related works in the intervening years. All of these works are based on approximation in value space and various forms of approximate policy iteration.

because some components of the state may be inaccessible for observation, the sensors used for measuring them may be inaccurate, or the cost of measuring them more accurately may be prohibitive.

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, solving a POMDP exactly is typically intractable, even though there are DP algorithms for doing so. Thus in practice, POMDP are solved approximately, except under very special circumstances.

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, which involves a different and more complicated state, called a *sufficient statistic*. Once this is done, we can state an exact DP algorithm that is defined over the space of the sufficient statistic. Roughly speaking, a sufficient statistic is a quantity that summarizes the content of the information available up to k for the purposes of optimal control. This statement can be made more precise, but we will not elaborate further in this book; see e.g., the DP textbook [Ber17a].

A common sufficient statistic is the *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 and all the controls applied 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 e.g., [AnM79], [KuV86], [Kri16], [ChC17]) and *particle filtering* (see e.g., [GSS93], [DoJ09], [Can16], [Kri16]).

Example 1.6.4 (Bidirectional Parking)

Let us consider a more complex version of the parking problem of Example 1.6.1. 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

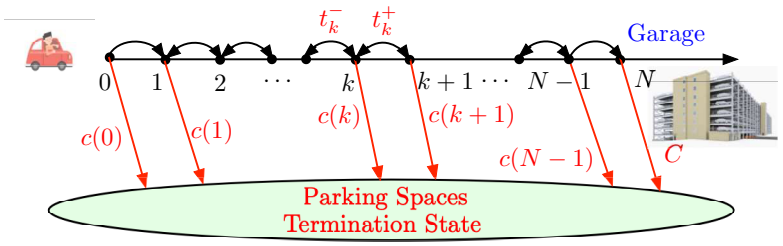

Figure 1.6.6 Cost structure and transitions of the bidirectional parking problem. The driver may park at space $k = 0, 1, \ldots, N-1$ at cost $c(k)$, if the space is free, can move to $k-1$ at cost t_k^- or can move to $k+1$ at cost t_k^+. At space N (the garage) the driver must park at cost C.

status of the spaces previously visited and may return to any of these spaces; see Fig. 1.6.6.

We 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 $1-\xi$.

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.

We can use the belief state as the basis of an exact DP algorithm for computing an optimal policy. This algorithm is typically intractable computationally, but it is conceptually useful, and it can form the starting point for approximations. It has the form

$$J_k^*(b_k) = \min_{u_k \in U_k} \left[\hat{g}_k(b_k, u_k) + E_{z_{k+1}} \Big\{ J_{k+1}^*\big(F_k(b_k, u_k, z_{k+1})\big) \;\Big|\; b_k, u_k \Big\} \right], \tag{1.73}$$

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

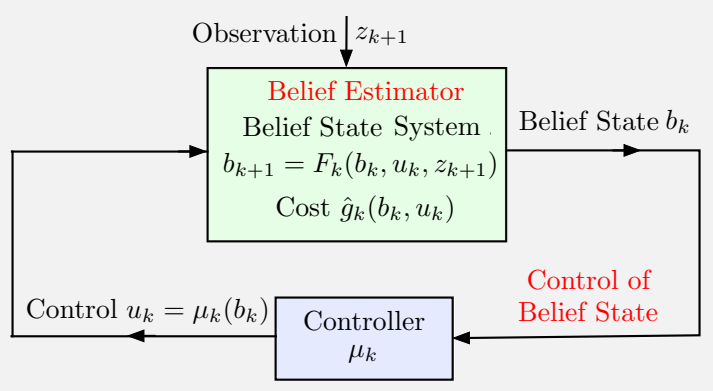

Figure 1.6.7 Schematic illustration of the view of an imperfect state information problem as one of perfect state information, whose state is the belief state b_k, i.e., the conditional probability distribution of x_k given all the observations up to time k. The observation z_{k+1} plays the role of the stochastic disturbance. The function F_k is a sequential estimator that updates the current belief state b_k.

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:

$$b_{k+1} = F_k(b_k, u_k, z_{k+1}). \tag{1.74}$$

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 $E_{z_{k+1}}\{\cdot \mid b_k, u_k\}$ 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

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

in Eq. (1.73) is a function of b_k and u_k.

The algorithm (1.73) is just the ordinary DP algorithm for the perfect state information problem shown in Fig. 1.6.7. It involves the system/belief estimator (1.74) 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 (1.73) will typically require an approximate implementation, using approximation in value space methods.

We refer to the textbook [Ber17a], Chapter 4, for a detailed derivation of the DP algorithm (1.73), and to the monograph [BeS78] for a mathematical treatment that applies to infinite-dimensional state and disturbance spaces as well.

An Alternative DP Algorithm for POMDP

The DP algorithm (1.73) is not the only one that can be used for POMDP. There is also an exact DP algorithm that operates in the space of information vectors I_k, defined by

$$I_k = \{z_0, u_0, \ldots, z_{k-1}, u_{k-1}, z_k\},$$

where z_k is the observation received at time k. This is another sufficient statistic, and hence an alternative to the belief state b_k. In particular, we can view I_k as a state of the POMDP, which evolves over time according to the equation

$$I_{k+1} = (I_k, z_{k+1}, u_k).$$

Denoting by $J_k^*(I_k)$ the optimal cost starting at information vector I_k at time k, the DP algorithm takes the form

$$J_k^*(I_k) = \min_{u_k \in U_k(x_k)} E_{w_k, z_{k+1}} \Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(I_k, z_{k+1}, u_k\big) \mid I_k, u_k \Big\}, \tag{1.75}$$

for $k = 0, \ldots, N-1$, with $J_N^*(I_N) = E\big\{g_N(x_N) \mid I_N\big\}$; see e.g., the DP textbook [Ber17a], Section 4.1.

A drawback of the preceding approach is that the information vector I_k is growing in size over time, thereby leading to a nonstationary system even in the case of an infinite horizon problem with a stationary system and cost function. This difficulty can be remedied in an approximation scheme that uses a finite history of the system (a fixed number of most recent observations) as state, thereby working effectively with a stationary finite-state system; see the paper by White and Scherer [WhS94]. In particular, this approach is used in large language models such as ChatGPT.

Finite-memory approximations for POMDP can be viewed within the context of feature-based approximation architectures, as we will discuss in Chapter 3 (see Example 3.1.6). Moreover, the finite-history scheme can be generalized through the concept of a *finite-state controller*; see the paper by Yu and Bertsekas [YuB08], which also addresses the issue of convergence of the approximation error to zero as the size of the finite-history or finite-state controller is increased.

1.6.7 Multiagent Problems and Multiagent Rollout

In this book, we will view a multiagent system as a collection of decision making entities, called *agents*, which aim to optimally achieve a common goal.† The agents accomplish this by collecting and exchanging information, and otherwise interacting with each other. The agents can be software programs or physical entities such as robots, and they may have different capabilities.

Among the generic challenges of efficient implementation of multiagent systems, one may note issues of limited communication and lack of fully shared information, due to factors such as limited bandwidth, noisy channels, and lack of synchronization. Another important generic issue is that as the number of agents increases, the size of the set of possible joint decisions of the agents increases exponentially, thereby complicating control selection by lookahead minimization. In this section, we will focus on ways to resolve this latter difficulty for problems where the agents fully share information, and in Section 2.9 we will address some of the challenges of problems where the agents may have some autonomy, and act without fully coordinating with each other.

For a mathematical formulation, let us consider the discounted infinite horizon problem and a special structure 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

$$U(x) = U^1(x) \times \cdots \times U^m(x), \tag{1.76}$$

where the sets $U^\ell(x)$ are given. This structure arises in applications involving distributed decision making by multiple agents; see Fig. 1.6.8.

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

$$\tilde{u} \in \arg\min_{u \in U(x)} E_w\Big\{g(x,u,w) + \alpha J_\mu\big(f(x,u,w)\big)\Big\}, \tag{1.77}$$

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 (1.76)]. 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.

† In a more general version of a multiagent system, which is outside our scope, the agents may have different goals, and act in their own self-interest.

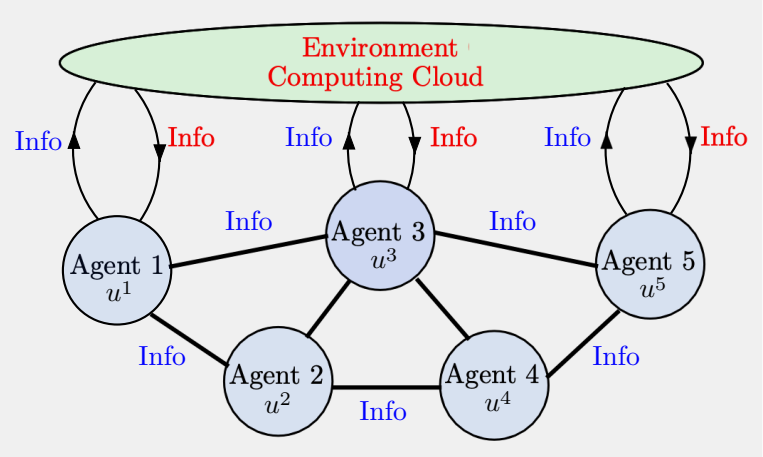

Figure 1.6.8 Schematic illustration of a multiagent problem. There are multiple "agents," and each agent $\ell = 1, \ldots, m$ controls its own decision variable u^ℓ. At each stage, agents exchange new information and also exchange information with the "environment," and then select their decision variables for the stage.

This potentially large computational overhead motivates a far more computationally efficient rollout algorithm, whereby the one-step lookahead minimization (1.77) is replaced by a sequence of m successive minimizations, *one-agent-at-a-time*, with the results incorporated into the subsequent minimizations. In particular, given a base policy $\mu = (\mu^1, \ldots, \mu^m)$, we perform at state x the sequence of minimizations

$$\tilde{\mu}^1(x) \in \arg\min_{u^1 \in U^1(x)} E_w\Big\{g\big(x, u^1, \mu^2(x), \ldots, \mu^m(x), w\big) + \alpha J_\mu\big(f(x, u^1, \mu^2(x), \ldots, \mu^m(x), w)\big)\Big\},$$

$$\tilde{\mu}^2(x) \in \arg\min_{u^2 \in U^2(x)} E_w\Big\{g\big(x, \tilde{\mu}^1(x), u^2, \mu^3(x) \ldots, \mu^m(x), w\big) + \alpha J_\mu\big(f(x, \tilde{\mu}^1(x), u^2, \mu^3(x), \ldots, \mu^m(x), w)\big)\Big\},$$

$$\cdots \qquad \cdots \qquad \cdots \qquad \cdots$$

$$\tilde{\mu}^m(x) \in \arg\min_{u^m \in U^m(x)} E_w\Big\{g\big(x, \tilde{\mu}^1(x), \tilde{\mu}^2(x), \ldots, \tilde{\mu}^{m-1}(x), u^m, w\big) + \alpha J_\mu\big(f(x, \tilde{\mu}^1(x), \tilde{\mu}^2(x), \ldots, \tilde{\mu}^{m-1}(x), u^m, w)\big)\Big\}.$$

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

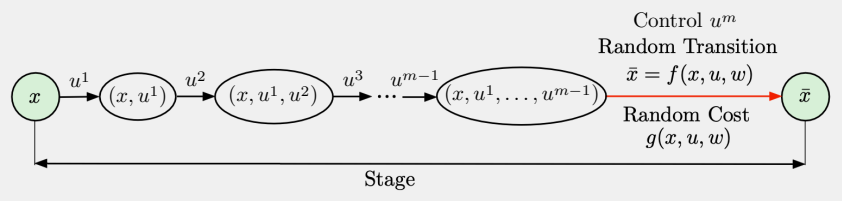

Figure 1.6.9 Equivalent formulation of the stochastic optimal control problem for the case where the control u consists of m components $u^1, u^2, \ldots, u^m$:

$$u = (u^1, \ldots, u^m) \in U(x) = U^1(x) \times \cdots \times U^m(x).$$

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

$$(x, u^1), (x, u^1, u^2), \ldots, (x, u^1, \ldots, u^{m-1}),$$

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.

$O(nm)$ 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 (1.77) are proportional to the number of controls in the set $U(x_k)$ and are independent of the size of the state space. This motivates a reformulation of the problem, first suggested in the neuro-dynamic programming 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 individual 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 a 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" $(x, u^1, \ldots, u^{m-1})$ marks the transition to the next state $\bar{x} = f(x, u, w)$, while incurring cost $g(x, u, w)$; see Fig. 1.6.9.

It is evident that this reformulated problem is equivalent to the origi-

nal, 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 $\mu = (\mu^1, \ldots, \mu^m)$ of the original problem, including a base policy in the context of rollout, is admissible for the reformulated problem, and has the same cost function for the original as well as the reformulated problem.

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 (1.77) is performed for just one state at each stage.

The major fact that can be proved about multiagent rollout (see Section 2.9 and the end-of-chapter references) is that it *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 $\mu = (\mu^1, \ldots, \mu^m)$, 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 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. Moreover, multiagent rollout becomes the starting point for related PI schemes that are well suited for distributed operation in contexts involving multiple autonomous decision makers; see Section 2.9, the book [Ber20a], the papers [Ber20b] and [BKB20], and the tutorial survey [Ber21a].

Example 1.6.5 (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. 1.6.10. The objective is for the spiders to catch all the flies as fast as possible.

During a stage, each fly moves to a some other position according to a given state-dependent 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 from 1 to m,

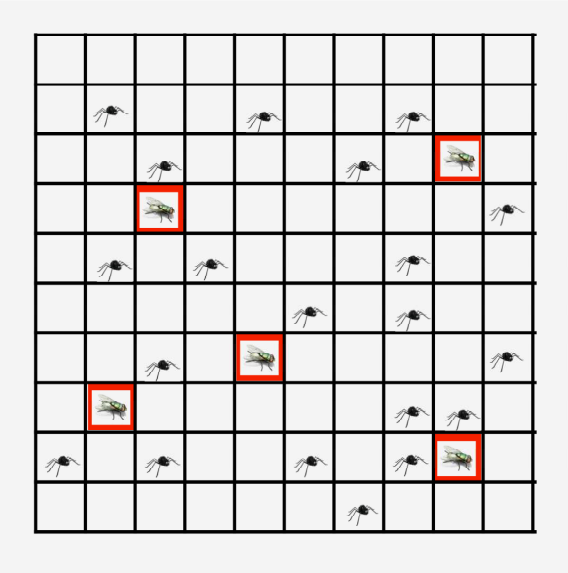

Figure 1.6.10 Illustration of a 2-dimensional spiders-and-fly problem with 20 spiders and 5 flies (cf. Example 1.6.5). The flies moves randomly, regardless of the position of the spiders. During a stage, each spider moves to a neighboring location or stays where it is, so there are 5 moves per spider (except for spiders at the edges of the grid). The total number of possible joint spiders moves is a little less than 5^{20}.

taking into account the current positions of the flies and the earlier moves of other spiders, and assuming that future moves will be chosen according to the base policy, which 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.

1.6.8 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.

It is important to note here 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.

In practice, however, it is common that the system parameters are either not known exactly or can change over time, and this introduces potentially enormous complications.† As an example consider our oversimplified cruise control system that we noted in Example 1.3.1 or its infinite horizon version. The state evolves according to

$$x_{k+1} = x_k + bu_k + w_k, \tag{1.78}$$

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.‡

In this section, we will briefly review some of the most commonly used approaches for dealing with unknown parameters in optimal control theory and practice. We should note also that unknown problem environments are

† The difficulties of decision and control within a changing environment 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 the 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.

an integral part of the artificial intelligence view of RL. In particular, to quote from the popular book by Sutton and Barto [SuB18], RL is viewed as "a computational approach to learning from interaction," and "learning from interaction with the environment is a foundational idea underlying nearly all theories of learning and intelligence."

The idea of learning from interaction with the environment is often 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, in a framework that is sometimes called *dual control*. This is one of the most challenging areas of stochastic optimal and suboptimal control, and has been studied intensively since the early 1960s.

Robust and Adaptive 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 investigate the performance of the current design for a suitable range of problem parameter values, and ensure that it is adequate for the entire range. This is sometimes called a *robust controller design*. For example, consider the oversimplified cruise control system of Eq. (1.78) with a linear controller of the form $\mu(x) = Lx$ for some scalar L. Then we check 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 ensure that b remains within that range during the system's operation.

The more general class of methods where the controller is modified in response to problem parameter changes is part of a broad field known as *adaptive control*, i.e., control that adapts to changing parameters. This is a rich methodology with many and diverse applications. Our discussion of adaptive control in this book will be limited. Let us just mention for the moment a simple time-honored adaptive control approach for continuous-state problems called *PID (Proportional-Integral-Derivative) control*, for which we refer to the control literature, including the books by Aström and Hagglund [AsH95], [AsH06], and the end-of-chapter references on adaptive control (also the discussion in Section 5.7 of the RL textbook [Ber19a]).

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.

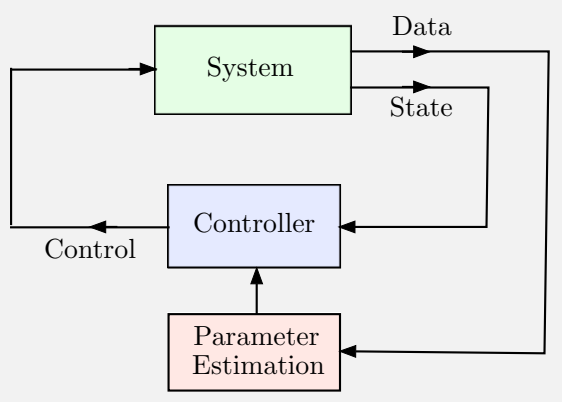

Figure 1.6.11 Schematic illustration of concurrent parameter estimation and system control. The system parameters are estimated on-line and the estimates are periodically passed on to the controller.

Dealing with Unknown Parameters Through System Identification

In PID control, no attempt is made to maintain a mathematical model and to track unknown model parameters as they change. An alternative and apparently reasonable form of suboptimal control 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 any system run in order to take into account subsequent changes of the parameters. Moreover, 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 introduced into the control process, whenever this is thought to be desirable; see Fig. 1.6.11.

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.

Example 1.6.6 (Parameter Identifiability Under Closed-Loop Control)

For a simple example, 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 control law 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 control law

$$\tilde{\mu}_k(x_k) = \gamma x_k \tag{1.79}$$

is used (γ is some scalar; for example, $\gamma = -\frac{\overline{a}}{\overline{b}}$, where $\overline{a}$ and $\overline{b}$ are some a priori estimates of a and b, respectively). At the end of the first phase, the control law 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 control law (1.79), 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 (1.79) 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 revise the controller using the newly estimated parameters in a variety of ways, in an 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 next.†

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 on-line replanning. We have discussed this approach in the context of rollout and multiagent rollout, where we stressed the importance of fast on-line policy improvement.

Let us assume that some problem parameters change and the current controller becomes aware of the change "instantly" (i.e., very quickly before the next stage begins). 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 1.6.5).

We thus assume away/ignore 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 the current policy used as the base policy is particularly attractive. Here the advantage of rollout is that it is simple and reliable. In particular, it does not require a complicated training procedure to revise the current policy, 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 current policy is used as the base policy for rollout, and the available controls at the current state are compared by a one-step or mutistep minimization, with cost function approximation provided by the base policy (cf. Fig. 1.6.12).

Note that *over time the base policy may also be revised* (on the basis of an unspecified rationale), in which case the rollout policy will be revised both in response to the changed current policy and in response to the

† 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 (for example each set of parameter values may correspond to a distinct 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.

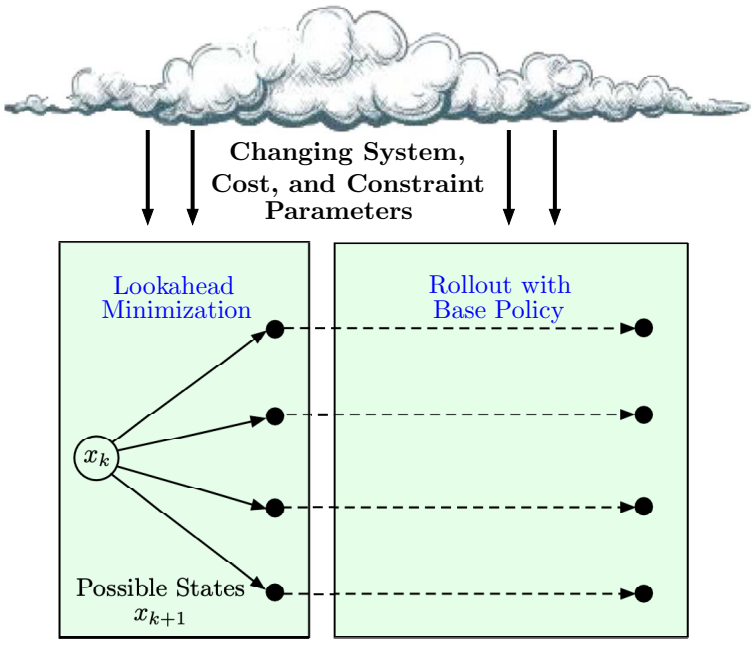

Figure 1.6.12 Schematic illustration of adaptive control by rollout. One-step lookahead is followed by simulation with the base policy, which stays fixed. The system, cost, and constraint parameters are changing over time, and the most recent values are incorporated into the lookahead minimization and rollout operations. 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.

changing parameters. This is necessary in particular when the constraints of the problem change.

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. Note, however, that accelerated/truncated versions of rollout, as well as parallel computation, can be used to meet this time constraint.

The following example considers 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 illustrate analytically how rollout with a 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. 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.

Example 1.6.7 (On-Line Replanning for Linear Quadratic Problems Based on Rollout)

Consider the deterministic undiscounted infinite horizon linear quadratic problem. 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

$$K = \frac{rK}{r + b^2 K} + 1. \tag{1.80}$$

The optimal policy has the form

$$\mu^*(x) = L^* x, \tag{1.81}$$

where

$$L^* = -\frac{bK^*}{r + b^2 K^*}. \tag{1.82}$$

As an example, consider the optimal policy that corresponds to the nominal problem parameters $b = 2$ and $r = 0.5$: this is the policy (1.81)-(1.82), with K obtained as the positive solution of the quadratic Riccati Eq. (1.80) for $b = 2$ and $r = 0.5$. In particular, we can verify that

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

From Eq. (1.82) we then obtain

$$L = -\frac{2 + \sqrt{6}}{5 + 2\sqrt{6}}. \tag{1.83}$$

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. (1.80).

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

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

where L is given by Eq. (1.83). From our earlier discussion, we have

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

(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 given earlier, we have

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

and

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

Figure 1.6.13 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. 1.6.13 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 quadratic convergence rate of Newton's method that underlies the rollout algorithm.

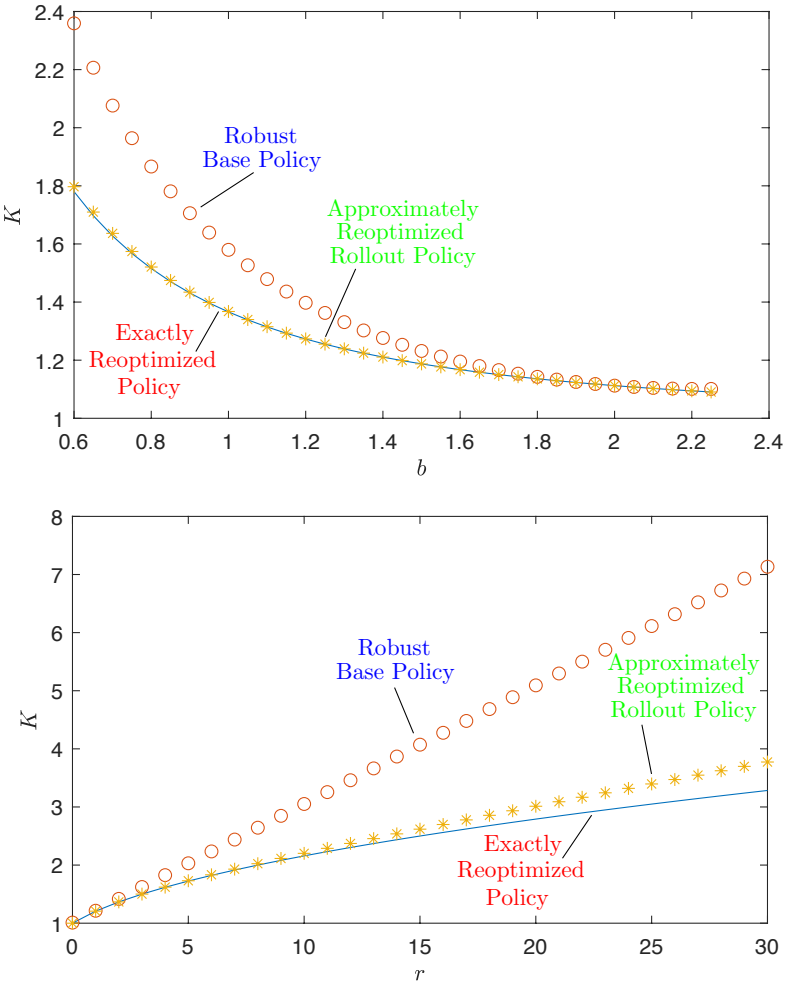

Figure 1.6.13 Illustration of adaptive control by rollout under changing problem parameters. The quadratic cost coefficients K^* (optimal, denoted by solid line), K_L (base policy, denoted by circles), and $\tilde{K}_L$ (rollout policy, denoted by asterisks) for the two cases where $r = 0.5$ and b varies, 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. (1.83)].

The rollout policy performance is very close to the one of the exactly reoptimized policy, while the base policy yields much worse performance. This is a consequence of the quadratic convergence rate of Newton's method that underlies rollout:

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

where for a given initial state, $\tilde{J}$ is the rollout performance, J^* is the optimal performance, and J is the base policy performance.

Adaptive Control as POMDP

The preceding adaptive control formulation strictly separates the dual objective of estimation and control: first parameter identification and then controller reoptimization (either exact or rollout-based). In an alternative adaptive control formulation, the parameter estimation and the application of control are done simultaneously, and indeed part of the control effort may be directed towards improving the quality of future estimation. This alternative (and more principled) approach is based on a view of adaptive control as a partially observed Markovian decision problem (POMDP) with a special structure. We will see in Section 2.11 that this approach is well-suited for approximation in value space schemes, including forms of rollout.

To describe briefly the adaptive control reformulation as POMDP, we introduce a system whose state consists of two components:

(a) A perfectly observed component x_k that evolves over time according to a discrete-time equation.

(b) A component θ which is unobserved but stays constant, and is estimated through the perfect observations of the component x_k.

We view θ as a parameter in the system equation that governs the evolution of x_k. Thus we have

$$x_{k+1} = f_k(x_k, \theta, u_k, w_k), \tag{1.84}$$

where u_k is the control at time k, selected from a set $U_k(x_k)$, and w_k is a random disturbance with given probability distribution that depends on (x_k, θ, u_k). For convenience, we will assume that θ can take one of m known values $\theta^1, \ldots, \theta^m$.

The a priori probability distribution of θ is given and is updated based on the observed values of the state components x_k and the applied controls u_k. In particular, the information vector

$$I_k = \{x_0, \ldots, x_k, u_0, \ldots, u_{k-1}\}$$

is available at time k, and is used to compute the conditional probabilities

$$b_{k,i} = P\{\theta = \theta^i \mid I_k\}, \qquad i = 1, \ldots, m.$$

These probabilities form a vector

$$b_k = (b_{k,1}, \ldots, b_{k,m}),$$

which together with the perfectly observed state x_k, form the pair (x_k, b_k), which is the *belief state* of the POMDP at time k. The overall control scheme takes the form illustrated in Fig. 1.6.14.

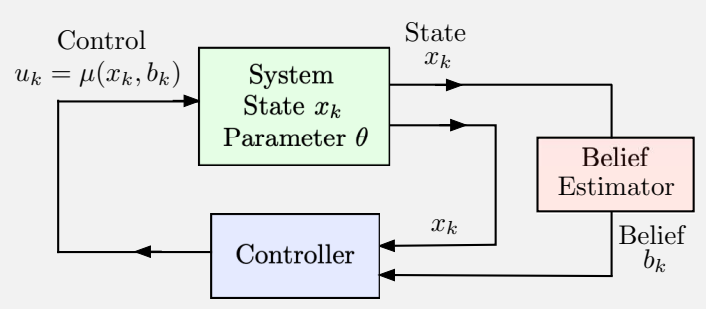

Figure 1.6.14 Schematic illustration of simultaneous control and belief estimation for the unknown system parameter θ. The control applied is a function of the current belief state (x_k, b_k), where b_k is the conditional probability distribution of θ given the observations accumulated up to time k (the current and past states $x_k, \ldots, x_0$, and the past controls $u_{k-1}, \ldots, u_0$).

As discussed in Section 1.6.4, an exact DP algorithm can be written for the equivalent POMDP, and this algorithm is suitable for the use of approximation in value space and rollout. We will describe this approach in some detail in Section 2.11. Related ideas will also be discussed in the context of Bayesian estimation and sequential estimation in Section 2.10.

Note that the case of a deterministic system

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

is particularly interesting, because we can then typically expect that the true parameter θ^* will be identified in a finite number of stages. The reason is that at each stage k, we are receiving a noiseless observation relating to θ, namely the state x_k. Once the true parameter θ^* is identified, the problem becomes one of perfect state information.

1.6.9 Model Predictive Control

In this section, we will provide a brief summary of the model predictive control (MPC) methodology for control system design, with a view towards its connection with approximation in value space and rollout schemes.† We will focus on classical control problems, where the objective is to keep the state of a deterministic system close to the origin of the state space (see Fig. 1.6.15). Another type of classical control problem is to keep the system

† An extensive overview of the connections of the conceptual framework of this book with model predictive and adaptive control is given in the author's paper [Ber24]. The corresponding video is a good supplement to the present section and can be found at https://www.youtube.com/watch?v=UeVs0Op-Ui4

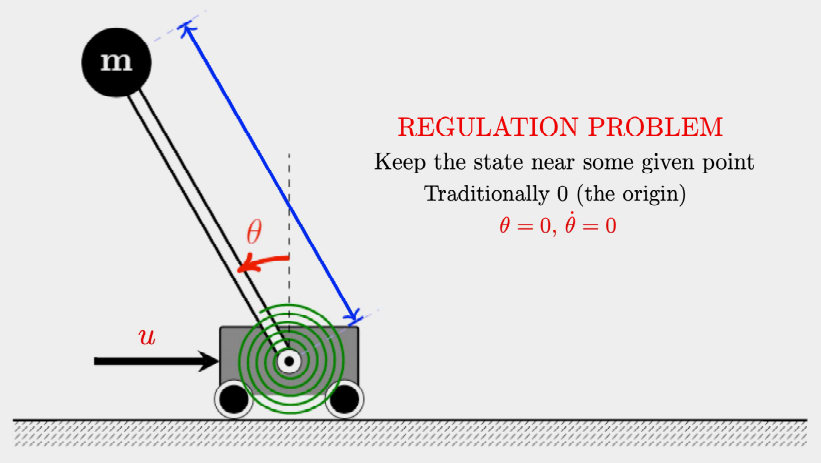

Figure 1.6.15 Illustration of a classical regulation problem, known as the "cart-pole problem" or "inverse pendulum problem." The state is the two-dimensional vector of angular position and angular velocity. We aim to keep the pole at the upright position (state equal to 0) by exerting horizontal force u on the cart.

close to a given trajectory (see Fig. 1.6.16). It can also be treated by forms of MPC, but will not be discussed in this book.

We discussed earlier the linear quadratic approach, whereby the system is represented by a linear model, the cost is quadratic in the state and the control, and there are no state and control constraints. The linear quadratic and other approaches based on state variable system representations and optimal control became popular, starting in the late 50s and early 60s. Unfortunately, however, the analytically convenient linear quadratic problem formulations are often not satisfactory. There are two main reasons for this:

(a) The system may be nonlinear, and it may be inappropriate to use for control purposes a model that is linearized around the desired point or trajectory. Moreover, some of the control variables may be naturally discrete, and this is incompatible with the linear system viewpoint.

(b) There may be control and/or state constraints, which are not handled adequately through quadratic penalty terms in the cost function. For example, the motion of a car may be constrained by the presence of obstacles and hardware limitations (see Fig. 1.6.16). The solution obtained from a linear quadratic model may not be suitable for such a problem, because quadratic penalties treat constraints "softly" and may produce trajectories that violate the constraints.

These inadequacies of the linear quadratic formulation have motivated MPC, which combines elements of several ideas that we have dis-

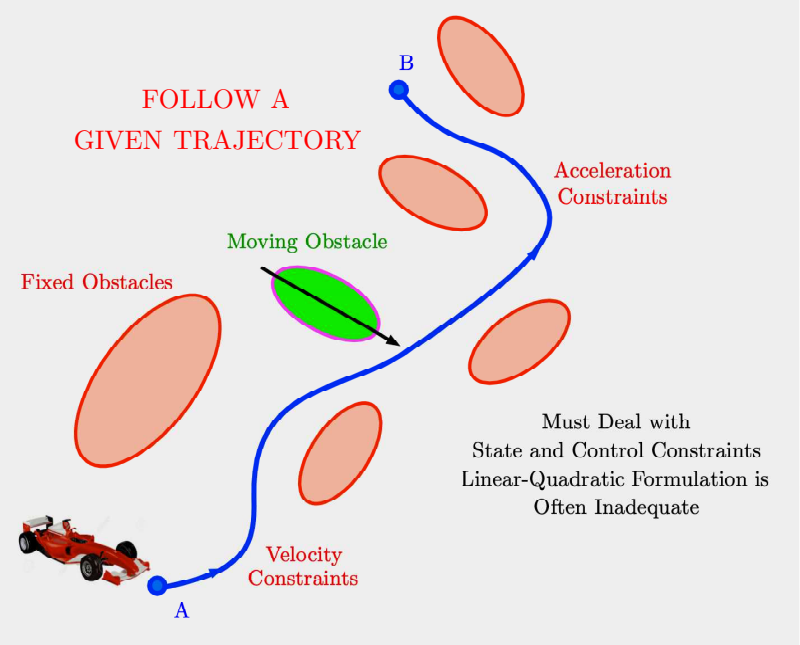

Figure 1.6.16 Illustration of constrained motion of a car from point A to point B. There are state (position/velocity) constraints, and control (acceleration) constraints. When there are mobile obstacles, the state constraints may change unpredictably, necessitating on-line replanning.

cussed so far, such as multistep lookahead, rollout with a base policy, and certainty equivalence. Aside from dealing adequately with state and control constraints, MPC is well-suited for on-line replanning, like all rollout methods.

Note that the ideas of MPC were developed independently of the approximate DP/RL methodology. However, the two fields are closely related, and there is much to be gained from understanding one field within the context of the other, as the subsequent development will aim to show. A major difference between MPC and finite-state stochastic control problems that are popular in the RL/artificial intelligence literature is that in MPC the state and control spaces are continuous/infinite, such as for example in self-driving cars, the control of aircraft and drones, or the operation of chemical processes. At the same time, at a fundamental level, this difference turns out to be inconsequential, because the key underlying framework for approximation in value space, which is based on Newton's method, is valid for both discrete and continuous state and control spaces.

In this section, we will primarily focus on the undiscounted infinite

horizon deterministic problem, which involves 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 initially consider 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, known as the *regulation problem*, 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. In an important variant of the problem, there are additional state constraints of the form $x_k \in X$, and there arises the issue of maintaining the state within X, not just at the present time but also in future times. We will address this issue later in this section.

The Classical Form of MPC - View as a Rollout Algorithm

We will first focus on a classical form of the MPC algorithm, proposed in the form given here by Keerthi and Gilbert [KeG88]. In this algorithm, at each encountered state x_k, we apply a control $\tilde{u}_k$ that is computed as follows; see Fig. 1.6.17:

(a) We solve an ℓ-stage optimal control problem involving the same cost function and the requirement that the state after ℓ steps is driven to 0, i.e., $x_{k+\ell} = 0$. This is the problem

$$\min_{u_t,\, t=k,\ldots,k+\ell-1} \sum_{t=k}^{k+\ell-1} g(x_t, u_t), \tag{1.85}$$

subject to the system equation constraints

$$x_{t+1} = f(x_t, u_t), \qquad t = k, \ldots, k+\ell-1, \tag{1.86}$$

the control constraints

$$u_t \in U(x_t), \qquad t = k, \ldots, k+\ell-1, \tag{1.87}$$

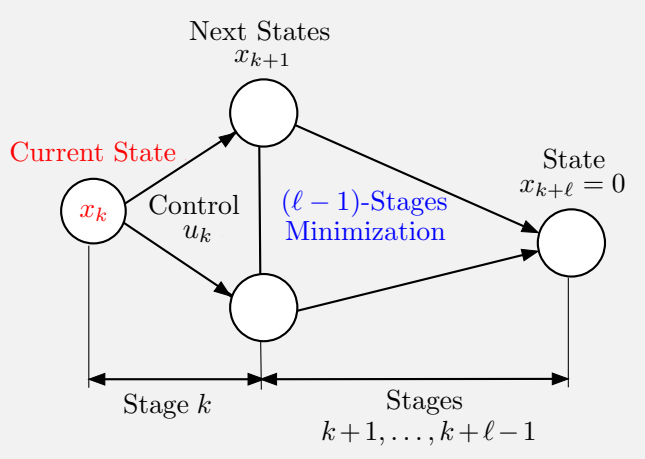

Figure 1.6.17 Illustration of the problem solved by a classical form of MPC at state x_k. We minimize the cost function over the next ℓ stages while imposing the requirement that $x_{k+\ell}=0$. We then apply the first control of the optimizing sequence. In the context of rollout, the minimization over 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.

and the terminal state constraint

$$x_{k+\ell}=0. \tag{1.88}$$

Here ℓ is an integer with $\ell>1$, which is chosen in some largely empirical way.

(b) If $\{\tilde{u}_k,\ldots,\tilde{u}_{k+\ell-1}\}$ is the optimal control sequence of this problem, we apply $\tilde{u}_k$ and we discard the other controls $\tilde{u}_{k+1},\ldots,\tilde{u}_{k+\ell-1}$.

(c) At the next stage, we repeat this process, once the next state x_{k+1} is revealed.

To make the connection of the preceding MPC algorithm with rollout, we note that *the one-step lookahead function $\tilde{J}$ implicitly used by MPC [cf. Eq. (1.85)] is the cost function of a certain stable base policy*. This is the policy 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 was first discussed in the author's paper [Ber05]. It is useful for making a connection with the approximate DP/RL, rollout, and its interpretation in terms of Newton's method. In particular, an important consequence is that *the MPC policy is stable*, since rollout with a stable base policy can be shown to yield a stable policy under very general conditions, as we have noted earlier for the special case of linear quadratic problems in Section 1.5; cf. Fig. 1.5.10.

Terminal Cost Approximation - Stability Issues

In a common variant of MPC, the requirement of driving the system state to 0 in ℓ steps in the ℓ-stage MPC problem (1.85), is replaced by a terminal cost $G(x_{k+\ell})$, which is positive everywhere except at 0. Thus at state x_k, we solve the problem

$$\min_{u_t,\, t=k,\ldots,k+\ell-1} \left[G(x_{k+\ell}) + \sum_{t=k}^{k+\ell-1} g(x_t, u_t) \right], \tag{1.89}$$

instead of problem (1.85) where we require that $x_{k+\ell} = 0$. This variant can be viewed as rollout with one-step lookahead, and a base policy, 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

$$G(x_{k+\ell}) + \sum_{t=k+1}^{k+\ell-1} g(x_t, u_t).$$

On the other hand, this MPC variant can also be viewed as approximation in value space with ℓ-step lookahead minimization and terminal cost approximation given by G. It can be interpreted in terms of a Newton step, as illustrated in Fig. 1.5.7 for the case of one-step lookahead, and in Fig. 1.5.8 for the case of multistep lookahead.

An important question is to choose the terminal cost approximation so that the resulting MPC controller is stable. Our discussion of Section 1.5 on the region of stability of approximation in value space schemes applies here. In particular, under the nonnegative cost assumption of this section, the MPC controller can be proved to be stable if a single value iteration (VI) starting from G produces a function that takes uniformly smaller values than G:

$$\min_{u \in U(x)} \Big\{ g(x,u) + G\big(f(x,u)\big) \Big\} \leq G(x), \qquad \text{for all } x. \tag{1.90}$$

Figure 1.6.18 provides a graphical illustration. It shows that this condition guarantees that successive iterates of value iteration, as implemented through multistep lookahead, lie within the region of stability, so that the policy produced by MPC is stable.

We also expect that as the length ℓ of the lookahead minimization is increased, the stability properties of the MPC controller are improved. In particular, *given $G \geq 0$, the resulting MPC controller is likely to be stable for ℓ sufficiently large*, since the VI algorithm ordinarily converges to J^*, which lies within the region of stability. Results of this type are known within the MPC framework under various conditions (see the papers by Mayne at al. [MRR00], Magni et al. [MDM01], the MPC book [RMD17], and the author's book [Ber20a], Section 3.1.2). Our discussion of stability in Section 1.5 is also relevant within this context; cf. Fig. 1.5.8.

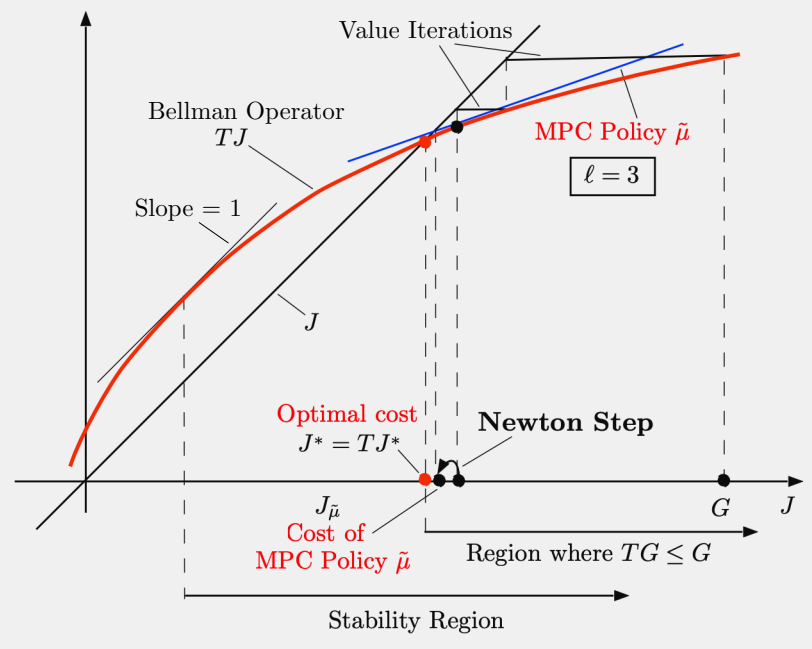

Figure 1.6.18 Illustration of the Bellman operator, defined by

$$(TJ)(x) = \min_{u \in U(x)} \Big\{ g(x,u) + J\big(f(x,u)\big) \Big\}, \qquad \text{for all } x.$$

The condition in (1.90) can be written compactly as $(TG)(x) \leq G(x)$ for all x. When satisfied by the terminal cost function G, it guarantees stability of the MPC policy $\tilde{\mu}$ with ℓ-step lookahead minimization. In this figure, $\ell = 3$.

In another variant of MPC, in addition to the terminal cost function approximation G, we use truncated rollout, which involves running some stable base policy μ for a number of steps m; see Fig. 1.6.19. This is quite similar to standard truncated rollout, except that the computational solution of the lookahead minimization problem (1.89) may become complicated when the control space is infinite. As discussed in Section 1.5, *increasing the length of the truncated rollout enlarges the region of stability of the MPC controller*. The reason is that by increasing the length of the truncated rollout, we push the start of the Newton step towards of the cost function J_μ of the stable policy, which lies within the region of stability. The base policy may also be used to address state constraints; see the papers by Rosolia and Borelli [RoB17], [RoB19], Li et al. [LJM21], and the discussions in the author's RL books [Ber20a], [Ber22a].

Finally, let us note that when faced with changing problem parameters, it is natural to consider on-line replanning as per our earlier adaptive

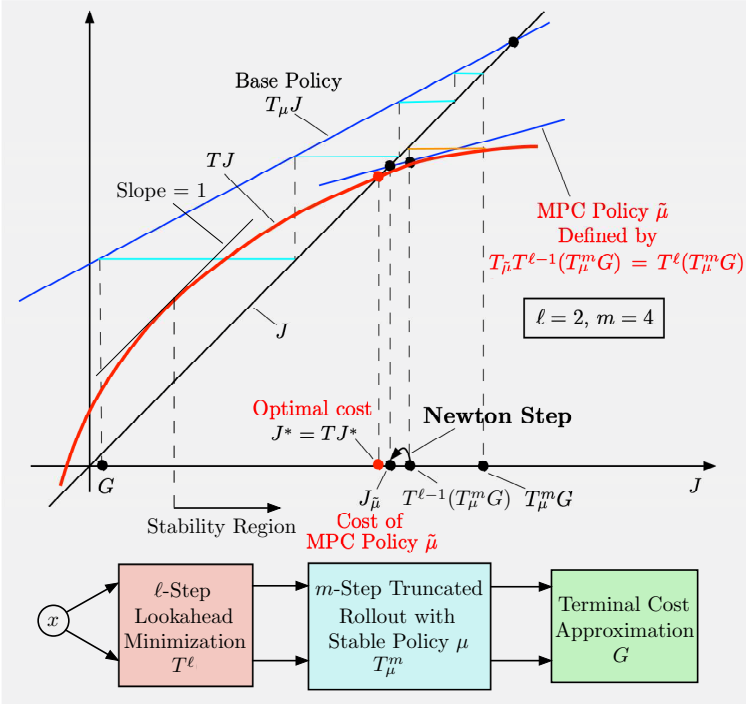

Figure 1.6.19 An MPC scheme with ℓ-step lookahead minimization, m-step truncated rollout with a stable base policy μ, and a terminal cost function approximation G, together with its interpretation as a Newton step. In this figure, $\ell = 2$ and $m = 4$. The truncated rollout with base policy μ consists of m value iterations with the Bellman operator corresponding to μ, which is given by

$$(T_\mu J)(x) = g\Big(x, \mu(x)\Big) + J\bigg(f\Big(x, \mu(x)\Big)\bigg).$$

Thus, truncated rollout applies m value iterations with base policy μ, starting with the function G and yielding the function $T_\mu^m G$. Then $\ell - 1$ value iterations are applied to $T_\mu^m G$ through the $(\ell - 1)$-step minimization. Finally, the Newton step is applied to

$$T^{\ell-1}(T_\mu^m G)$$

to yield the cost function of the MPC policy $\tilde{\mu}$. As m increases, the starting point for the Newton step moves closer to J_μ, which lies within the region of stability.

control discussion. In this context, 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.

State Constraints, Invariant Sets, and Off-Line Training

Our discussion so far has skirted a major issue in MPC, which is that there may be additional state constraints of the form $x_k \in X$, for all k, where X is some subset of the true state space. Indeed much of the original work on MPC was motivated by control problems with state constraints, imposed by the physics of the problem, which could not be handled effectively with the nice unconstrained framework of the linear quadratic problem that we have discussed in Section 1.5.

To deal with additional state constraints of the form $x_k \in X$, where X is some subset of the state space, the MPC problem to be solved at the kth stage [cf. Eq. (1.89)] must be modified. Assuming that the current state x_k belongs to the constraint set X, the MPC problem should take the form

$$\min_{u_t,\, t=k,\ldots,k+\ell-1} \left[G(x_{k+\ell}) + \sum_{t=k}^{k+\ell-1} g(x_t, u_t) \right], \tag{1.91}$$

subject to the control constraints

$$u_t \in U(x_t), \qquad t = k, \ldots, k+\ell-1, \tag{1.92}$$

and the state constraints

$$x_t \in X, \qquad t = k+1, \ldots, k+\ell. \tag{1.93}$$

The control $\tilde{u}_k$ thus obtained will generate a state

$$x_{k+1} = f(x_k, \tilde{u}_k)$$

that will belong to X, and similarly the entire state trajectory thus generated will satisfy the state constraint $x_t \in X$ for all t, assuming that the initial state does.

However, there is an important difficulty with the preceding MPC scheme, namely *there is no guarantee that the problem (1.91)-(1.93) has a feasible solution for all initial states $x_k \in X$*. Here is a simple example.

Example 1.6.8 (State Constraints in MPC)

Consider the scalar system

$$x_{k+1} = 2x_k + u_k,$$

with control constraint

$$|u_k| \leq 1,$$

and state constraints of the form $x_k \in X$, for all k, where

$$X = \big\{ x_k \mid |x_k| \leq \beta \big\}. \tag{1.94}$$

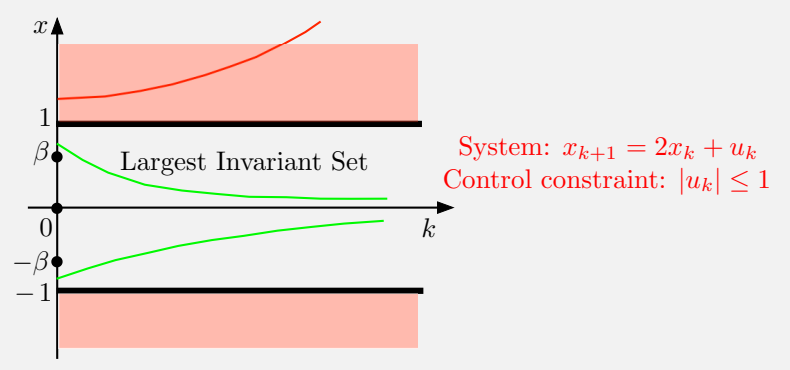

Figure 1.6.20 Illustration of invariance of a state constraint set X. Here the sets of the form $X = \{x_k \mid |x_k| \leq \beta\}$ are invariant for $\beta \leq 1$. For $\beta = 1$, we obtain the largest invariant set (the one that contains all other invariant sets). The figure shows some state trajectories produced by MPC. Note that starting with an initial condition x_0 with $|x_0| > 1$ (or $|x_0| < 1$) the closed-loop system obtained by MPC is unstable (or stable, respectively); cf. the red and green trajectories shown.

Then if $\beta > 1$, the state constraint cannot be satisfied for all initial states $x_0 \in X$. In particular, if we take $x_0 = \beta$, then $2x_0 > 2$ and $x_1 = 2x_0 + u_0$ will satisfy $x_1 > x_0 = \beta$ for any value of u_0 with $|u_0| \leq 1$. Similarly the entire sequence of states $\{x_k\}$ generated by any set of feasible controls will satisfy

$$x_{k+1} > x_k \quad \text{for all } k, \qquad x_k \uparrow \infty.$$

The state constraint can be satisfied only for initial states x_0 in the set $\hat{X}$ given by

$$\hat{X} = \{x_k \mid |x_k| \leq 1\};$$

see Fig. 1.6.20, which also illustrates the trajectories generated by the MPC scheme of Eq. (1.89), which does not involve state constraints.

The preceding example illustrates a fundamental point in state-constrained MPC: *the state constraint set X must be invariant in the sense that starting from any one of its points x_k there must exist a control $u_k \in U(x_k)$ for which the next state $x_{k+1} = f(x_k, u_k)$ must belong to X.* Mathematically, X is invariant if

$$\text{for every } x \in X, \text{ there exists } u \in U(x) \text{ such that } f(x, u) \in X.$$

In particular, it can be seen that the set X of Eq. (1.94) is invariant if and only if $\beta \leq 1$.

Given an MPC calculation of the form (1.91)-(1.93), we must make sure that the set X is invariant, or else it should be replaced by an invariant subset $\hat{X} \subset X$. Then the MPC calculation (1.91)-(1.93) will be feasible provided the initial state x_0 belongs to $\hat{X}$.

This brings up the question of how we compute an invariant subset of a given constraint set, which is typically an off-line calculation that cannot be performed during on-line play. It turns out that given X there exists a largest possible invariant subset of X, which can be computed in the limit with an algorithm that resembles value iteration. In particular, starting with $X_0 = X$, we obtain a nested sequence of subsets through the recursion

$$X_{k+1} = \big\{x \in X_k \mid f(x,u) \text{ belongs to } X_k \text{ for some } u \in U(x)\big\}, \quad k \geq 0. \tag{1.95}$$

Clearly, we have $X_{k+1} \subset X_k$ for all k, and under mild conditions it can be shown that the intersection set $\hat{X} = \cap_{k=0}^{\infty} X_k$, is the largest invariant subset of X; see the author's PhD thesis [Ber71] and subsequent paper [Ber72a], which introduced the concept of invariance and its use in satisfying state constraints in control over a finite and an infinite horizon.†

To illustrate, in the preceding example, the sequence of value iterates (1.95) starting with the set $X_0 = \{x \mid |x| \leq \beta\}$, where $\beta > 1$, is given by

$$X_k = \{x \mid |x| \leq \beta_k\}, \quad \text{with } \beta_0 = \beta \text{ and } \beta_{k+1} = \tfrac{\beta_k + 1}{2} \text{ for all } k \geq 0.$$

It can be seen that we have $\beta_{k+1} < \beta_k$ for all k and $\beta_k \downarrow 1$, so the intersection $\hat{X} = \cap_{k=0}^{\infty} X_k$ yields the largest invariant set $\hat{X} = \big\{x_k \mid |x_k| \leq 1\big\}$.

There are several methods to compute invariant subsets of constraint sets X, for which we refer to the aforementioned author's work and the MPC literature; see e.g., the book by Rawlings, Mayne, and Diehl [RMD17], and the surveys by Mayne [May14], and Houska, Muller, and Villanueva [HMV24], which give additional references. An important point is that the computation of an invariant subset of the given constraint set X must be done off-line with one of several available algorithmic approaches, thus becoming part of the off-line training phase, along with the terminal cost function G. A relatively simple possibility is to compute an invariant subset $\hat{X}$ that corresponds to some nominal policy $\hat{\mu}$ [i.e., starting from any point $x \in \hat{X}$, the state $f\big(x, \hat{\mu}(x)\big)$ belongs to $\hat{X}$]. Such an invariant subset may be obtained by some form of simulation using the policy $\hat{\mu}$. Moreover, $\hat{\mu}$ can also be used for truncated rollout and also provide a terminal cost function approximation.

Once an off-line training process provides the invariant set $\hat{X}$, the terminal cost function G, and potentially a base policy for truncated rollout, MPC becomes an on-line play algorithm for which our earlier discussion applies. Note, however, that in an adaptive control context, where a model is estimated on-line as it is changing, it may be difficult to recompute on-line an invariant set that can be used to enforce the state constraints of the

† The term used in [Ber71] and [Ber72a] is *reachability of a target tube* $\{X, X, \ldots\}$, which is synonymous to invariance of X.

problem. This is particularly so if the state constraints change themselves as part of the changing problem data. In such cases it is often preferable to replace the state constraints with penalty or barrier functions as part of the cost per stage. This approach has received a lot of attention recently, using what is known as *control barrier functions* and *control Lyapunov functions*. We refer to the literature for related accounts.

Stochastic MPC by Certainty Equivalence

Let us finally note that while in this section we have focused on deterministic problems, there are variants of MPC, which include the treatment of uncertainty. The books and papers cited earlier contain several ideas along these lines; see e.g. the books by Kouvaritakis and Cannon [KoC16], Rawlings, Mayne, and Diehl [RMD17], and the survey by Mesbah [Mes16].

In this connection, it is also worth mentioning the *certainty equivalence approach* that we discussed briefly earlier. In particular, upon reaching state x_k we may perform the MPC calculations after replacing the uncertain quantities $w_{k+1}, w_{k+2}, \ldots$ with deterministic quantities $\overline{w}_{k+1}, \overline{w}_{k+2}, \ldots$, *while allowing for the stochastic character of the disturbance w_k of just the current stage k.* Note that only the first step of this MPC calculation is stochastic. Thus the calculation needed per stage is not much more difficult than the one for deterministic problems, while still implementing a Newton step for solving the associated Bellman equation; see our earlier discussion, and also Section 2.5.3 of the RL book [Ber19a], Section 3.2 of the book [Ber22a], and the MPC overview paper [Ber24].

1.7 REINFORCEMENT LEARNING AND DECISION/CONTROL

The current state of RL has greatly benefited from the cross-fertilization of ideas from decision and control, and from artificial intelligence; see Fig. 1.7.1. The strong connections between these two fields are now widely recognized. Still, however, there are cultural differences, including the traditional reliance on mathematical analysis for the decision and control field, and the emphasis on challenging problem implementations in the artificial intelligence field. Moreover, 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).

1.7.1 Differences in Terminology

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 AI or the optimal control terminology, we provide a list of terms commonly used in RL, and their optimal control counterparts.

(a) **Environment** = System.

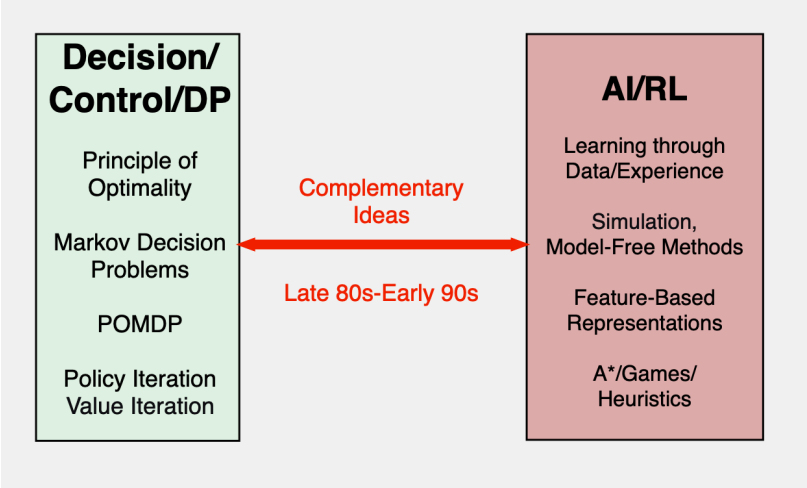

Figure 1.7.1 A schematic illustration of the synergy of ideas between artificial intelligence on one hand, and decision and control on the other.

(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.

(v) **Backup** = Applying the DP operator at some state.

(w) **Sweep** = Applying the DP operator at all states.

(x) **Greedy policy with respect to a cost function** J = Minimizing policy in the DP expression defined by J.

(y) **Afterstate** = Post-decision state.

(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.

1.7.2 Differences in Notation

Unfortunately, the confusion caused by differing terminology has been further compounded by the use of inconsistent notations across various sources. This book adheres to the "standard" notation that emerged during the Bellman/Pontryagin optimal control era; see e.g., the books by Athans and Falb [AtF66], Bellman [Bel67], and Bryson and Ho [BrH75]. This notation is consistent with the author's other DP books and is the most appropriate for a unified treatment of the subject, which simultaneously addresses discrete and continuous spaces problems.

A summary of the most prominently used symbols in our notational system is as follows:

(a) x: state (also i for finite-state systems).

(b) u: control.

(c) w: stochastic disturbance.

(d) J: cost function.

(e) f: system function. For deterministic systems,

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

and for stochastic systems,

$$x_{k+1} = f(x_k, u_k, w_k).$$

Also f_k in place of f for time-varying systems.

(f) g: cost per stage [$g(x,u)$ for deterministic systems, and $g(x,u,w)$ for stochastic systems; also g_k in place of g for time-varying systems].

(g) $p_{xy}(u)$: transition probability from state x to state y under control u in finite-state systems [also $p_{ij}(u)$].

(h) α: discount factor in discounted problems.

The x-u-J notation is standard in deterministic optimal control textbooks (e.g., the classical books [AtF66] and [BrH75], noted earlier, as well as the more recent books by Stengel [Ste94], Kirk [Kir04], and Liberzon [Lib11]). The symbols f (system function) and g (cost per stage) are also widely used in both early and later optimal control literature (unfortunately the more natural symbol "c" has not been used much in place of "g" for the cost per stage).

The notations i (state) and $p_{ij}(u)$ (transition probability) are common in the discrete-state Markov decision process (MDP) and operations research literature. Sometimes the alternative notation $p(j \mid i, u)$ is used for the transition probabilities.

In the artificial intelligence literature, the focus is primarily on finite-state MDPs, particularly discounted and stochastic shortest path infinite horizon problems. 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. While this notation is well-suited to finite-state problems, it is not ideal for continuous spaces models, which are of major interest in this book. The reason is that it requires the use of transition probability distributions defined over continuous spaces, and leads to more complex and less intuitive mathematics. Moreover, for deterministic problems, which lack a probabilistic component, the transition probability notation becomes cumbersome and unnecessary.

1.7.3 A Few Words about Machine Learning and Mathematical Optimization

Machine learning and optimization are closely intertwined fields, sharing similar mathematical models and computational algorithms.† However, they differ in their cultures and application contexts, so it is worth reflecting on their similarities and differences.

Machine learning can be broadly categorized into three main types of methods, all of which involve the collection and use of data in some form:

† Both fields are also closely connected to the field of statistical analysis. However, in this section, we will not focus on this connection, as it is less relevant to the content of this book.

(a) *Supervised learning*: Here a dataset of many input-output pairs (also called labeled data) is collected. An optimization algorithm is used to create a parametrized function that fits well the data, as well as make accurate predictions on new, unseen data. Supervised learning problems are typically formulated as optimization problems, examples of which we will see in Chapter 3. A common algorithmic approach is to use a gradient-type algorithm to minimize a loss function that measures the difference between the actual outputs of the dataset and the predicted outputs of the parametrized model.

(b) *Unsupervised learning*: Here the dataset is "unlabeled" in the sense that the data are not separated into input and matching output pairs. Unsupervised learning algorithms aim to identify patterns or structures within the data, which is useful for tasks like clustering, dimensionality reduction, and density estimation. The objective is to extract meaningful insights from the data. Some unsupervised learning methods can be related to DP, but the connection is not strong. Generally speaking, unsupervised learning does not seem to align well with the types of sequential decision making applications of this book.

(c) *Reinforcement learning*: RL differs in an important way from supervised and unsupervised learning. *It does not use a dataset as a starting point*. Instead, it generates data on-line or off-line as dictated by the needs of the optimization algorithm it uses, be it multistep lookahead minimization, approximate policy iteration and rollout, or approximation in policy space.†

Another type of machine learning approach, which relates to DP/RL methods, is *semi-supervised learning*. It involves training a model using a dataset containing both labeled and unlabeled data. Here, some initial labeled data are sequentially augmented with unlabeled data, with the aim of constructing an "informative" data set that enhances machine learning tasks such as classification. This approach lies between supervised learning (which requires all data to be labeled) and unsupervised learning (which works with exclusively unlabeled data). Semi-supervised learning is related to the field of *active learning*, where DP-like methods are used to augment sequentially the labeled set; see e.g., the monograph by Zhu and Goldberg [ZhG22], the survey by Van Engelen and Hoos [VaH20], and the illustrative application papers by Marchesoni-Acland et al. [MMK23], and Bhusal, Miller, and Merkurjev [BMM24].

Optimization problems and algorithms on the other hand may or may not involve the collection and use of data. They involve data only in the context of special applications, most of which are related to machine learning. In theoretical terms, optimization problems are categorized in terms

† A variant of RL called *offline RL* or *batch RL*, starts from a historical dataset, and does not explore the environment to collect new data.

of their mathematical structure, which is the primary determinant of the suitability of particular types of methods for their solution. In particular, it is common to distinguish between *static optimization problems* and *dynamic optimization problems*. The latter problems involve sequential decision making, with feedback between decisions, while the former problems involve a single decision.

Stochastic problems with perfect or imperfect state observations are dynamic (unless they involve open-loop decision making without the use of any feedback), and they require the use of DP for their optimal solution. Deterministic problems can be formulated as static, but they can also be formulated as dynamic for reasons of algorithmic expediency. In this case, the decision making process is (sometimes artificially) broken down into stages, as is often done in this book for discrete optimization and other contexts.

Another important categorization of optimization problems is based on whether their search space is *discrete* or is *continuous*. Discrete problems include deterministic problems such as integer and combinatorial optimization problems, and can be addressed by formal methods of integer programming as well as by DP. These problems tend to be challenging, so they are often addressed (suboptimally) with the use of heuristics. Continuous problems are usually addressed with very different methods, which are based on calculus and convexity, such as Lagrange multiplier theory and duality, and the computational machinery of linear, nonlinear, and convex programming. Some discrete problems, particularly those that involve graphs (such as matching, transportation, and transshipment problems), can be addressed using network optimization methods that rely on linear programming and duality concepts. Hybrid problems, which combine discrete and continuous variables, usually require discrete optimization techniques but can also benefit from convex duality methods, which are fundamentally continuous.

The DP methodology, generally speaking, applies to just about any kind of optimization problem, deterministic or stochastic, static or dynamic, discrete or continuous, *as long as it is formulated as a sequential decision problem*, in the manner described in Sections 1.2-1.4. In terms of algorithmic structure, DP differs significantly from other optimization techniques, particularly those based on calculus and convexity. Notably, DP can handle both discrete and continuous problems and is not concerned with local minima, focusing instead on finding global minima.

Notice a qualitative difference between optimization and machine learning: *the former is mostly organized around mathematical structures and the analysis of the foundational issues of the corresponding algorithms, while the latter is mostly organized around how data is collected, used, and analyzed, often with a strong emphasis on statistical issues*. This is an important distinction, which affects profoundly the perspectives of researchers in the two fields.

Relations Between RL and DP Methodologies

When comparing the RL and DP methodologies, it is important to recognize that they are fundamentally connected by their shared focus on sequential decision making. Thus, any problem that can be addressed by DP can, in principle, also be addressed by RL, and vice versa.

One may argue that the RL algorithmic methodology is broader than that of DP. It includes the use of optimization algorithms of the gradient descent and random search type, simulation-based methodologies, statistical methods of sampling and performance evaluation, and neural network design and training ideas. However, methods of this type have also been considered in DP-related research and applications for many years.

In the artificial intelligence view of RL, a machine learns through trial and error by interacting with an environment.† In practical terms, this is more or less the same as what DP aims to do, but in RL there is often an emphasis on the presence of uncertainty and exploration of the environment. In the decision, control, and optimization community, there is a lot of interest in using RL methods to address intractable problems, including deterministic discrete/integer optimization, which need not involve data collection, interaction with the environment, uncertainty, and learning (adaptive control is the only decision and control problem type, where uncertainty and exploration arise in a significant way).

In terms of applications, DP was originally developed in the 1950s and 1960s as part of the then emerging methodologies of operations research and optimal control. These methodologies are now mature and provide important tools and perspectives, as well as a rich variety of applications, such as robotics, autonomous transportation, and aerospace, which can benefit from the use of RL. Moreover, DP has been used in a broad range of applications in industrial engineering, operations research, economics, and finance, so these applications can also benefit from the use of RL methods and perspectives.

At the same time, RL and machine learning have ushered opportunities for the application of DP techniques in new domains, such as machine translation, image recognition, knowledge representation, database organization, large language models, and automated planning, where they can have a significant practical impact. We may also add that RL has brought into the field of sequential decision making a fresh and ambitious spirit that has made possible the solution of problems thought to be well outside the capabilities of DP. In particular, before the connections between RL and DP were recognized, large dimensional problems, like those involving a Euclidean state space of even moderate dimension, or POMDP problems, were considered totally intractable with the DP methodology.

† A common description is that "the machine learns sequentially how to make decisions that maximize a reward signal, based on the feedback received from the environment."

The Use of Mathematics in Optimization and Machine Learning

Let us now discuss some differences between the research cultures of optimization and machine learning, as they pertain to the use of mathematics. In optimization, the emphasis is often on general purpose methods that offer broad and mathematically rigorous performance guarantees, for a wide variety of problems. In particular, it is widely believed that a solid mathematical foundation for a given optimization methodology enhances its reliability and clarifies the boundaries of its applicability. Furthermore, it is recognized that formulating practical problems and matching them to the right algorithms is greatly enhanced by one's understanding of the mathematical structure of the underlying optimization methodology.

Machine learning research includes important lines of analysis that have a strongly mathematical character, particularly relating to theoretical computer science, complexity theory, and statistical analysis. At the same time, in machine learning there are eminently useful algorithmic structures, such as neural networks, large language models, and image generative models, which are not well-understood mathematically and defy to a large extent mathematical analysis.† This can add to a perception that focusing on rigorous mathematics, as opposed to practical implementation, may be a low payoff investment in many practical machine learning contexts.

Moreover, the starting point in machine learning is often a specific dataset or a specialized type of training problem (e.g., language translation or image recognition). The priority is to find a method that works well for that specific dataset or problem, even if it is not generalizable to others. Thus specialized approximation architectures, implementation techniques, and heuristics, which perform well for the given problem and dataset type, may be perfectly acceptable in a machine learning context, even if they do not provide rigorous and generally applicable performance guarantees.

In conclusion, both optimization and machine learning involve mathematical models and rigorous analysis in important ways, and often overlap in the techniques and tools that they use, as well as in the practical applications that they address. However, depending on the problem at hand, there may be differences in the emphasis and priority placed on mathematical analysis, insight, and generality versus practical effectiveness and problem-specific efficiency. This can lead to some tension, as different fields may not fully appreciate each other's perspective.

---

† As an illustration, the paper by He et al., "Deep Residual Learning for Image Recognition," published in Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, 2016, has been cited over 162,000 times as of May 2023, and contains only two equations. The famous neural network architecture paper by Vaswani et al., "Attention is all you Need," published in NIPS, 2017, which laid the foundation for GPT, has been cited over 73,000 times as of May 2023, and contains only six equations.

1.8 NOTES, SOURCES, AND EXERCISES

We will now summarize this chapter and describe how it can be used flexibly as a foundation for a few different courses. We will also provide a selective overview of the DP and RL literature, and also give a few exercises that have been used in ASU classes.

Chapter Summary

In this chapter, we have aimed to provide an overview of the approximate DP/RL landscape, which can serve as the foundation for a deeper in-class development of other RL topics. In particular, we have described in varying levels of depth the following:

(a) The algorithmic foundation of exact DP in all its major forms: deterministic and stochastic, discrete and continuous, finite and infinite horizon.

(b) Approximation in value space with one-step and multistep lookahead, the workhorse of RL, which underlies its major success stories, including AlphaZero. We contrasted approximation in value space with approximation in policy space, and discussed how the two may be combined.

(c) The important division between off-line training and on-line play in the context of approximation in value space. We highlighted how their synergy can be intuitively explained in terms of Newton's method.

(d) The fundamental methods of policy iteration and rollout, the former being primarily an off-line method, and the latter being primarily a less ambitious on-line method. Both methods and their variants bear close relation to Newton's method and draw their effectiveness from this relation.

(e) Some major models with a broad range of applications, such as discrete optimization, POMDP, multiagent problems, adaptive control, and model predictive control. We delineated their principal characteristics and the major RL implementation issues within their contexts.

(f) The use of function approximation, which has been a recurring theme in our presentation. We have touched upon some of the principal schemes for approximation, e.g., neural networks and feature-based architectures.

One of the principal aims of this chapter was to provide a foundational platform for a range of RL courses that explore at a deeper level various algorithmic methodologies, such as:

(1) Rollout and policy iteration.

(2) Neural networks and other approximation architectures for off-line training.

(3) Aggregation, which can be used for cost function approximation in the context of approximation in value space.

(4) A broader discussion of sequential decision making in contexts involving changing system parameters, sequential estimation, and simultaneous system identification and control.

(5) Stochastic algorithms, such as temporal difference methods and Q-learning, which can be used for off-line policy evaluation in the context of approximate policy iteration.

(6) Sampling methods to collect data for off-line training in the context of cost and policy approximations.

(7) Statistical estimates and efficiency enhancements of various sampling methods used in simulation-based schemes. This includes confidence intervals and computational complexity estimates.

(8) On-line methods for specially structured contexts, including problems of the multi-armed bandit type.

(9) Simulation-based algorithms for approximation in policy space, including policy gradient and random search methods.

(10) A deeper exploration of control system design methodologies such as model predictive control and adaptive control, and their applications in robotics and automated transportation.

In our course we have focused selectively on the methodologies (1)-(4), with a limited coverage of (9) in Section 3.5. In a different course, other choices from the above list may be made, by building on the content of the present chapter.

Notes and Sources for Individual Sections

In the literature review that follows, we will focus primarily on textbooks, research monographs, and broad surveys, which supplement our discussions, present related viewpoints, and collectively provide a guide to the literature. Inevitably, our selection reflects a certain cultural bias and an overemphasis on sources that are familiar to the author and aligned in style with this book (including the author's own works). We acknowledge in advance that this may lead to omissions of research references that fall outside our own understanding and perspective on the field, and we apologize for any such exclusions.

Sections 1.1-1.4: Our discussion of exact DP in this chapter has been brief since our focus in this book will be on approximate DP and RL. For a more comprehensive treatment of finite-horizon exact DP and its applications to both discrete and continuous space problems, the author's DP textbook [Ber17a] provides an extensive overview, using notation and style consistent with this book. The books by Puterman [Put94] and by the

author [Ber12] provide detailed (but substantially different) treatments of infinite horizon finite-state stochastic DP problems. The book [Ber12] also covers continuous/infinite state and control spaces problems, including the linear quadratic problems that we have discussed for one-dimensional problems in this chapter. Continuous spaces problems present special analytical and computational challenges, which are at the forefront of research of the RL methodology. The author's 1976 DP textbook [Ber76] was the first to develop DP within a framework that allows arbitrary state, control, and disturbance spaces.

Some of the more complex mathematical aspects of exact DP were addressed in the monograph by Bertsekas and Shreve [BeS78], particularly the probabilistic/measure-theoretic issues associated with stochastic optimal control, including partial state information problems. This monograph provides an extensive treatment of these issues. The followup work by Huizhen Yu and the author [YuB15] resolves the special measurability issues that relate to policy iteration, and provides further analysis relating to the convergence of value iteration. The second volume of the author's DP book [Ber12], Appendix A, includes an accessible summary introduction of the measure-theoretic framework of the book [BeS78].† In the RL literature, the mathematical difficulties around measurability are usually neglected (as they are in this book), and this is fine because they do not play an important role in applications. Moreover, measurability issues do not arise for problems involving finite or countably infinite state and control spaces. We note, however, that there are quite a few published works in RL as well as exact DP, which purport to address measurability issues with a mathematical narrative that is either confusing or plain incorrect.

---

† The rigorous mathematical theory of stochastic optimal control, including the development of an appropriate measure-theoretic framework, originated in the 60s and 70s, with the work of Blackwell and other mathematicians. It culminated in the monograph [BeS78], which provides the now "standard" framework, based on the formalism of Borel spaces, lower semianalytic functions, and universally measurable policies. This development involves daunting mathematical complications, which stem, among others, from the observation that when a Borel measurable function $F(x,u)$, of the two variables x and u, is minimized with respect to u, the resulting function $G(x) = \min_u F(x,u)$ need not be Borel measurable (it belongs to the broader class of lower semianalytic functions; see [BeS78]). Moreover, even if the minimum is attained by several policies μ, i.e., $G(x) = F\big(x,\mu(x)\big)$ for all x, it is possible that none of these μ is Borel measurable (however, there does exist a minimizing policy that belongs to the broader class of universally measurable policies). Thus, starting with a Borel measurability framework for cost functions and policies, we quickly get outside that framework when executing DP algorithms, such as value and policy iteration. The broader framework of universal measurability, introduced in [BeS78], is required to correct this deficiency, in the absence of additional (fairly strong) assumptions.

The third edition of the author's abstract DP monograph [Ber22b], expands on the original 2013 first edition, and aims at a unified development of the core theory and algorithms of total cost sequential decision problems. It addresses simultaneously stochastic, minimax, game, risk-sensitive, and other DP problems, through the use of abstract DP operators (or Bellman operators as we call them here). The idea 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. Abstract DP ideas are also useful for visualizations and interpretations of RL methods using the Newton method formalism that we have discussed somewhat briefly in this book in the context of linear quadratic problems.

Approximation in value space, rollout, and policy iteration 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. Moreover, rollout is reliable, easy to implement, and can be used in conjunction with on-line replanning. It is also compatible with new and exciting technologies such as transformer networks and large language models (see Section 2.3.7).

As we have noted, rollout with a given base policy is simply the first iteration of the policy iteration algorithm starting from the base policy. Truncated rollout can be interpreted as an "optimistic" form of a single policy iteration, whereby a policy is evaluated inexactly, by using a limited number of value iterations; see the books [Ber20a], [Ber22a].‡

† 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 to make move decisions for both sides."

‡ Truncated rollout was also proposed in the context of backgammon in the paper [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

Policy iteration, which can be seen as repeated 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 more detail later. Note that rollout does not require any off-line training, once the base policy is available; this is its principal advantage over policy iteration.

Section 1.5: There is a vast literature on linear quadratic problems. The connection of policy iteration with Newton's method within this context and its quadratic convergence rate was first derived by Kleinman [Kle68] for continuous-time linear quadratic problems (the corresponding discrete-time result was given by Hewer [Hew71]). For followup work, which relates to policy iteration with approximations, see Feitzinger, Hylla, and Sachs [FHS09], and Hylla [Hyl11].

The general relation of approximation in value space with Newton's method, beyond policy iteration, and its connections with MPC and adaptive control was first presented in the author's book [Ber20a], the papers [Ber21b], [Ber22c], and in the book [Ber22a], which contains an extensive discussion. This relation provides the starting point for an in-depth understanding of the synergy between the off-line training and the on-line play components of the approximation in value space architecture, including the role of multistep lookahead in enhancing the starting point of the Newton step. The monograph [Ber22a] also provides analysis of variants of Newton's method applied to nondifferentiable fixed point problems, such as the ones arising in the context of Bellman's equation (which involves nondifferentiabilities in finite-control space problems, among others).

Note that in approximation in value space, we are applying Newton's method to the solution of a system of equations (the Bellman equation). This context has no connection with the "gradient descent" methods that are popular for the solution of special types of optimization problems in RL, arising for example in neural network training problems (see Chapter 3). In particular, there are no gradient descent methods that can be used for the solution of systems of equations such as the Bellman equation. There are, however, "first order" deterministic algorithms such as the Gauss-Seidel and Jacobi methods (and stochastic asynchronous extensions) that can be applied to the solution of systems of equations with special structure, including Bellman equations. Such methods include various Q-learning

---

the actual outcome. The truncated rollout algorithm requires much less CPU time, due to two factors: First, there are potentially many 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.

algorithms, which are discussed in the neuro-dynamic programming book by Bertsekas and Tsitsiklis [BeT89], as well as the recent book by Meyn [Mey22]. While these methods can be useful, they are typically much slower than Newton's method and have limited utility in the context of on-line play.

Section 1.6: Many applications of DP are discussed in the 1st volume of the author's DP book [Ber17a]. This book also covers a broad variety of state augmentation and problem reformulation techniques, including the mathematics of how problems with imperfect state information can be transformed to perfect state information problems. In Section 1.6 we have aimed to provide an overview, with an emphasis on the use of approximations. In what follows we provide some related historical notes.

Multiagent problems: This subject has a long history (Marschak [Mar55], Radner [Rad62], Witsenhausen [Wit68], [Wit71a], [Wit71b]), and was researched extensively in the 70s; see the review paper by Ho [Ho80] and the references cited there. The names used 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. In addition to the aforementioned works, surveys of multiagent sequential decision making from an RL perspective were given by Busoniu, Babuska, and De Schutter [BBD08], [BBD10b].

The term "multiagent" has been used with various meanings in the literature. Some authors emphasize scenarios where agents lack common information when making their decisions, leading to sequential decision problems with "nonclassical information patterns." These problems are particularly complex because they cannot be solved using exact DP techniques. Other authors focus on situations where the agents are "weakly" coupled through the system equation, the cost function, or the constraints. They consider methods that exploit the weak coupling to address the problem with (suboptimal) decoupled computations.

Agent-by-agent minimization in multiagent approximation in value space and rollout was proposed in the author's paper [Ber19c], which also discusses extensions to infinite horizon policy iteration algorithms, and explores connections with the concept of person-by-person optimality from team theory; see also the textbook [Ber20a], the papers [Ber19d], [Ber20b]. The papers by Bhattacharya et al. [BKB20], Garces et al. [GBG22], and Weber at al. [WGP23] present computational studies with challenging problems, where several of the multiagent algorithmic ideas were adapted, tested, and validated. These papers consider large-scale multi-robot and

vehicle routing problems, involving partial state information, and explore some of the attendant implementation issues, including autonomous multiagent rollout, through the use of policy neural networks and other precomputed signaling policies. They also compare the performance of these multiagent methods against alternative approaches, including those based on policy gradient techniques.

A different form of distributed computation and multiagent optimization, where each agent has a partial/local model of the system within part of the state space and relies on aggregate information from other agents for DP computations, is proposed in the author's DP book [Ber12], Section 6.5.4; see also Section 3.5.8 of the present book.

Adaptive control: The research on adaptive control has a long history and its literature is very extensive; see the books by Aström and Wittenmark [AsW94], Aström and Hagglund [AsH06], Bodson [Bod20], Goodwin and Sin [GoS84], Ioannou and Sun [IoS96], Jiang and Jiang [JiJ17], Krstic, Kanellakopoulos, and Kokotovic [KKK95], Kumar and Varaiya [KuV86], Liu, et al. [LWW17], Lavretsky and Wise [LaW13], Narendra and Annaswamy [NaA12], Sastry and Bodson [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 time series models, to extremum-seeking methods, to simulation-based RL techniques, etc.

The ideas of PID control have been applied widely to adaptive and robust control contexts, and have a long history; see the books by Aström and Hagglund [AsH95], [AsH06], which provide many references. 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" in 1922 [Min22].

The DP framework for adaptive control was introduced in a series of papers by Feldbaum, starting in 1960 with [Fel60], under the name *dual control theory*. These papers emphasized the division of effort between system estimation and control, now more commonly referred to as the *exploration-exploitation tradeoff*. In the last paper of the series [Fel63], Feldbaum prophetically concluded as follows: "At the present time, the most important problem for the immediate future is the development of approximate solution methods for dual control theory problems, the formulation of sub-optimal strategies, the determination of the numerical value of risk in quasi-optimal systems and its comparison with the value of risk in existing systems."

The research on problems involving unknown models and using data for model identification 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 on "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. This is more or less the same as the classical problems of dual and adaptive control that have been discussed since the 60s from a control theory perspective.

The formulation of adaptive and dual control problems as POMDP (cf. Section 2.11) is classical. The use of rollout within this context was first suggested in the author's book [Ber22a], Section 6.7.

Model predictive control: The literature on the theory and applications of MPC is voluminous. Some early widely cited papers are Clarke, Mohtadi, and Tuffs [CMT87a], [CMT87b], and Keerthi and Gilbert [KeG88]. For surveys, which give many of the early references, see Morari and Lee [MoL99], Mayne et al. [MRR00], and Findeisen et al. [FIA03], and for a more recent review, see Mayne [May14]. 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 connections between MPC, approximation in value space and rollout were discussed in the author's surveys [Ber05a] and [Ber24].

Reinforcement Learning Sources

The first DP/RL 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 is culturally different and 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 these books deal with finite-state Markovian decision models and use a transition probability notation, as they do not address continuous spaces problems, which are one of the major focal points of 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 methods, Chang, Fu, Hu, and Marcus [CFH13] (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. The book by Meyn [Mey22] focuses on the connections of RL and optimal control, similar to the present book, but is more mathematically oriented, and treats stochastic problems and algorithms in more detail.

There are also several books that, while not exclusively focused on DP and/or RL, touch upon some of the topics of the present 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 a discussion of both exact DP, and approximate DP/RL methods. The textbooks by Kouvaritakis and Cannon [KoC16], Borrelli, Bemporad, and Morari [BBM17], and Rawlings, Mayne, and Diehl [RMD17] collectively provide a comprehensive view of the MPC methodology. The book by Lattimore and Szepesvari [LaS20] is focused on multiarmed bandit methods. The book by Brandimarte [Bra21] is a tutorial introduction to DP/RL that emphasizes operations research applications and includes MATLAB codes. The book by Hardt and Recht [HaR21] focuses on broader subjects of machine learning but covers selectively approximate DP and RL topics as well.

The present book is similar in style, terminology, and notation to the author's recent textbooks [Ber19a] (Reinforcement Learning and Optimal Control), [Ber20a] (Rollout and Policy Iteration), [Ber22a] (Lessons from AlphaZero), and the 3rd edition of the abstract DP monograph [Ber22b], which collectively provide a fairly comprehensive and more mathematical account of the subject. In particular, the book [Ber19a] includes a broader coverage of approximation in value space methods, including certainty equivalent control and aggregation methods. It also addresses ap-

proximation in policy space in greater detail than the present book. The book [Ber20a] focuses more closely on rollout, policy iteration, and multiagent problems, and introduced the connection of approximation in value space with Newton's method. The book [Ber22a] focuses primarily on this connection, relying on analysis first provided in the book [Ber20a] and the paper [Ber22c]. The abstract DP monograph [Ber22b] (a 3rd edition of the original 2013 1st edition) is an advanced treatment of exact DP, which provides the mathematical framework of Bellman operators that are central for some of the Newton method visualizations presented in the present book and in the books [Ber20a], [Ber22a].

In addition to textbooks, there are many surveys and short research monographs relating to our subject, which are rapidly multiplying in number. Influential early surveys were written, from an artificial intelligence viewpoint, by Barto, Bradtke, and Singh [BBS95] (which dealt with the methodologies of real-time DP and its antecedent, real-time heuristic search [Kor90], and the use of asynchronous DP ideas [Ber82], [Ber83], [BeT89] within their context), and by Kaelbling, Littman, and Moore [KLM96] (which focused on general principles of RL). The volume by White and Sofge [WhS92] also contains surveys describing early work in the field.

Several overview papers in the volume by Si, Barto, Powell, and Wunsch [SBP04] describe some approximation methods that we will not be covering in much detail in this book: linear programming approaches (De Farias [DeF04]), large-scale resource allocation methods (Powell and Van Roy [PoV04]), and deterministic optimal control approaches (Ferrari and Stengel [FeS04], and Si, Yang, and Liu [SYL04]). Updated accounts of these and other related topics are given in the survey collections by Lewis, Liu, and Lendaris [LLL08], and Lewis and Liu [LeL13].

Recent extended surveys and short monographs are Borkar [Bor09] (a methodological point of view that explores connections with other Monte Carlo schemes), Lewis and Vrabie [LeV09] (a control theory point of view), Szepesvari [Sze10] (which discusses approximation in value space from a RL point of view), Deisenroth, Neumann, and Peters [DNP11], and Grondman et al. [GBL12] (which focus on policy gradient methods), Browne et al. [BPW12] (which focuses on Monte Carlo Tree Search), Mausam and Kolobov [MaK12] (which deals with Markovian decision problems from an artificial intelligence viewpoint), Geffner and Bonet [GeB13] (which deals with problems in search and automated planning), Schmidhuber [Sch15], Arulkumaran et al. [ADB17], Li [Li17], Busoniu et al. [BDT18], and Caterini and Chang [CaC18] (which deal with reinforcement learning schemes that are based on the use of deep neural networks), Recht [Rec18a] (which discusses continuous spaces optimal control), and the author's [Ber05a] (which focuses on rollout algorithms and MPC), [Ber11a] (which focuses on approximate policy iteration), [Ber18a] (which focuses on aggregation methods), [Ber20b] (which focuses on multiagent problems), and [Ber24] (which focuses on the relations between RL and MPC).

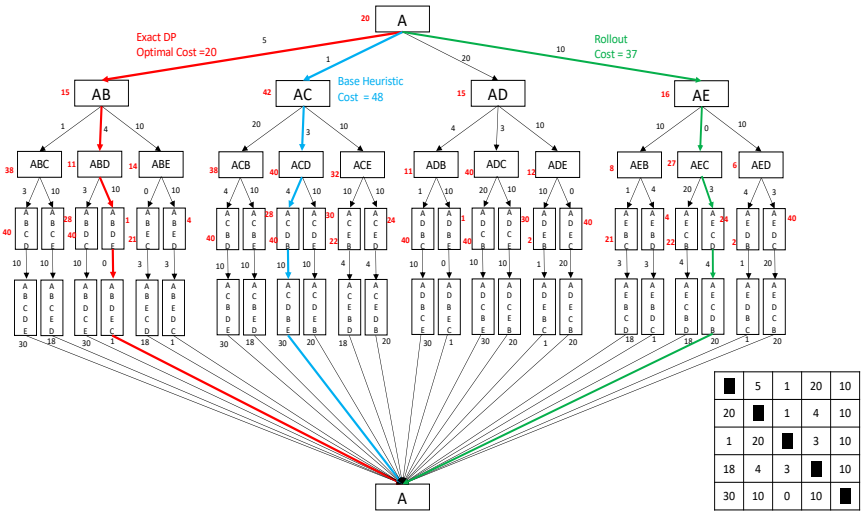

Figure 1.8.1 Solution of parts (a), (b), and (c) of Exercise 1.1. A 5-city traveling salesman problem illustration of rollout with the nearest neighbor base heuristic.

E X E R C I S E S

1.1 (Computational Exercise - Traveling Salesman Problem)

Consider a modified version of the four-city traveling salesman problem of Example 1.2.3, where there is a fifth city E. The intercity travel costs are shown in Fig. 1.8.1, which also gives the solutions to parts (a), (b), and (c).

(a) Use exact DP with starting city A to verify that the optimal tour is ABDECA with cost 20.

(b) Verify that the nearest neighbor heuristic starting with city A generates the tour ACDBEA with cost 48.

(c) Apply rollout with one-step lookahead minimization, using as base heuristic the nearest neighbor heuristic. Show that it generates the tour AECDBA with cost 37.
Illustration of the algorithm: At city A, the nearest neighbor heuristic generates the tour ACDBEA with cost 48, as per part (b). At city A, the rollout algorithm considers the four options of moving to cities B, C, D, E, or equivalently to states AB, AC, AD, AE, and it computes the nearest neighbor-generated tours corresponding to each of these states. These tours are ABCDEA with cost 49, ACDBEA with cost 48, ADCEBA with cost

63, and AECDBA with cost 37. The tour AECDBA has the least cost, so the rollout algorithm moves to city E or equivalently to state AE.

At AE, the rollout algorithm considers the three options of moving to cities B, C, D, or equivalently to states AEB, AEC, AED, and it computes the nearest neighbor-generated tours corresponding to each of these states. These tours are AEBCDA with cost 42, AECDBA with cost 37, AEDCBA with cost 63. The tour AECDBA has the least cost, so the rollout algorithm moves to city C or equivalently to state AEC.

At AEC, the rollout algorithm considers the two options of moving to cities B, D, and compares the nearest neighbor-generated tours corresponding to each of these. These tours are AECBDA with cost 52 and AECDBA with cost 37. The tour AECDBA has the least cost, so the rollout algorithm moves to city D or equivalently to state AECD. Then the rollout algorithm has only one option and generates the tour AECDBA with cost 37.

(d) Apply rollout with two-step lookahead minimization, using as base heuristic the nearest neighbor heuristic. This rollout algorithm operates as follows. For $k = 1, 2, 3$, it starts with a k-city partial tour, it generates every possible two-city addition to this tour, uses the nearest neighbor heuristic to complete the tour, and selects as next city to add to the k-city partial tour the city that corresponds to the best tour thus obtained (only one city is added to the current tour at each step of the algorithm, not two). Show that this algorithm generates the optimal tour.

(e) Estimate roughly the complexity of the computations in parts (a), (b), (c), and (d), assuming a generic N-city traveling salesman problem. *Answer*: The exact DP algorithm requires $O(N^N)$ computation, since there are

$$(N-1)+(N-1)(N-2)+\cdots+(N-1)(N-2)\cdots 2+(N-1)(N-2)\cdots 2\cdot 1$$

arcs in the DP graph to consider, and this number can be estimated as $O(N^N)$. The nearest neighbor heuristic that starts at city A performs $O(N)$ comparisons at each of N stages, so it requires $O(N^2)$ computation. The rollout algorithm at stage k runs the nearest neighbor heuristic $N-k$ times, so it must run the heuristic $O(N^2)$ times for a total computation of $O(N^4)$. Thus the rollout algorithm's complexity involves a low order polynomial increase over the complexity of the base heuristic, something that is generally true for practical discrete optimization problems. Note that even though this may represent a substantial increase in computation over the base heuristic, it is a potentially enormous improvement over the complexity of the exact DP algorithm.

1.2 (Computational Exercise - Linear Quadratic Problem)

In this problem we focus on the one-dimensional linear quadratic problem of Section 1.5 and the interpretation of approximation space as a Newton step for solving the Riccati equation. Consider the undiscounted linear quadratic problem with parameters $a = 2$, $b = 1$, $q = 1$, $r = 5$. For this problem:

(a) Plot and solve graphically the Riccati equation as in Fig. 1.5.1.

(b) Plot and solve graphically the Riccati equation corresponding to the linear policy $\mu(x) = -(3/2)x$.

(c) Plot graphically the numerical solution of the Riccati equation by value iteration as in Fig. 1.5.3, using a starting point $K_0 < K^*$ and a starting point $K_0 > K^*$.

(d) Interpret graphically approximation in value space with one-step, two-step, and three-step lookahead as a Newton step in the manner of Figs. 1.5.7 and 1.5.8. Use cost function approximations $\tilde{K}x^2$ with $\tilde{K} < K^*$ and $\tilde{K} > K^*$. What is the region of stability, i.e., the set of $\tilde{K}$ for which approximation in value space produces a stable policy under one-step, two-step, and three-step lookahead.

(e) Plot the performance error $|K_{\tilde{\mu}} - K^*|$ as a function of $|\tilde{K} - K^*|$ for one-step, two-step, and three-step lookahead approximation in value space.

(f) Plot graphical interpretations of rollout and truncated rollout in the manner of Figs. 1.5.10 and 1.5.11 using a stable starting linear policy of your choice.

1.3 (Computational Exercise - Spiders and Flies)

Consider the spiders and flies problem of Example 1.6.5 with two differences: the five flies stay still (rather than moving randomly), and there are only two spiders, both of which start at the fourth square from the right at the top row of the grid of Fig. 1.6.10. The base policy 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 1.6.5, and compare its performance with the one of the ordinary rollout algorithm, and with the one of the base policy. This problem is also discussed in Section 2.9.

1.4 (Computational Exercise - Exercising an Option)

This exercise deals with a computational comparison of the optimal policy, a heuristic policy, and on-line approximation in value space using the heuristic policy, in the context of a problem that involves the timing of the sale of a stock.

An investor has the option to sell a given amount of stock at any one of N time periods. The initial price of the stock is an integer x_0. The price x_k, if it is positive and it is less than a given positive integer value $\bar{x}$, it evolves according to

$$x_{k+1} = \begin{cases} x_k + 1 & \text{with probability } p^+, \\ x_k & \text{with probability } 1 - p^+ - p^-, \\ x_k - 1 & \text{with probability } p^-, \end{cases}$$

where p^+ and p^- have known values with

$$0 < p^- \leq p^+, \qquad p^+ + p^- < 1.$$

If $x_k = 0$, then x_{k+1} moves to 1 with probability p^+, and stays unchanged at 0 with probability $1 - p^+$. If $x_k = \bar{x}$, then x_{k+1} moves to $\bar{x} - 1$ with probability p^-, and stays unchanged at $\bar{x}$ with probability $1 - p^-$.

At each period $k = 0, \ldots, N-1$ for which the stock has not yet been sold, the investor (with knowledge of the current price x_k), can either sell the stock at the current price x_k or postpone the sale for a future period. If the stock has not been sold at any of the periods $k = 0, \ldots, N-1$, it must be sold at period N at price x_N. The investor wants to maximize the expected value of the sale. For the following computations, use reasonable values of your choice for N, p^+, p^-, $\bar{x}$, and x_0 (you should choose x_0 between 0 and $\bar{x}$). You are encouraged to experiment with different sets of values. A set of values that you may try first is

$$N = 14, \quad x_0 = 3, \quad \bar{x} = 7, \quad p^+ = p^- = 0.25.$$

(a) Formulate the problem as a finite horizon DP problem by identifying the state, control, and disturbance spaces, the system equation, the cost function, and the probability distribution of the disturbance. Write the corresponding exact DP algorithm, and use it to compute the optimal policy and the optimal cost as a function of x_0.
Solution: The optimal reward-to-go is generated by the following DP algorithm:

$$J_N^*(x_N) = x_N, \tag{1.96}$$

and for $k = 0, \ldots, N-1$, if $x_k = 0$, then

$$J_k^*(0) = p^+ J_{k+1}^*(1) + (1 - p^+) J_{k+1}^*(0), \tag{1.97}$$

if $x_k = \bar{x}$, then

$$J_k^*(\bar{x}) = \bar{x}, \tag{1.98}$$

(since the price cannot go higher than $\bar{x}$, once at $\bar{x}$, but can go lower), and if $0 < x_k < \bar{x}$, then

$$J_k^*(x_k) = \max\left\{x_k,\, p^+ J_{k+1}^*(x_k{+}1) + (1{-}p^+{-}p^-) J_{k+1}^*(x_k) + p^- J_{k+1}^*(x_k{-}1)\right\}. \tag{1.99}$$

The optimal policy is to sell at $x_k = 1, \ldots, \bar{x}-1$, if x_k attains the maximum in the above equation, and not to sell otherwise. When $x_k = 0$, it is optimal not to sell, while when $x_k = \bar{x}$, it is optimal to sell.

The values of $J_k^*(x_k)$ and the optimal policy are tabulated as shown in Fig. 1.8.2. For this figure, all the calculations are done for the following special case:

$$N = 10, \quad x_0 = 2, \quad \bar{x} = 10, \quad p^+ = p^- = 0.25.$$

These values are also used for parts (b) and (c). However, you are asked to solve the problem for different values as noted earlier. Note that for the problem to have an interesting solution, the problem data must be chosen so that the problem's policies are materially affected by the presence of the upper and lower bounds on the price x_k. As an example consider the case where

$$N = 10, \quad x_0 = 20, \quad \bar{x} = 40, \quad p^+ = p^- = 0.25.$$

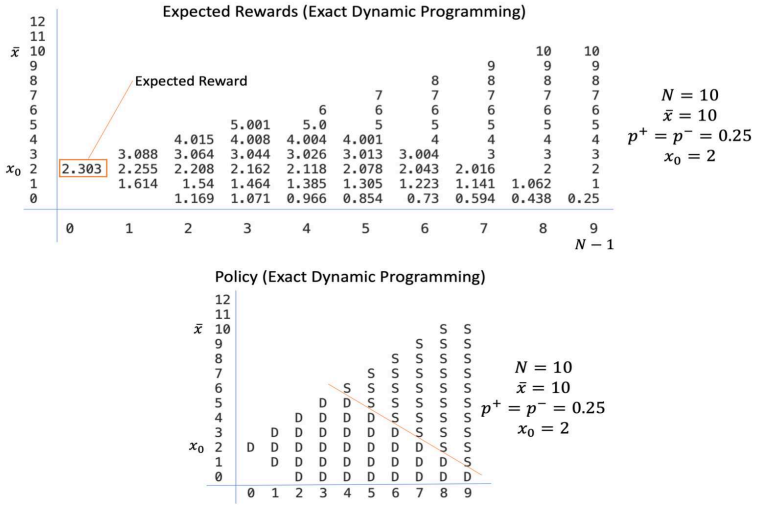

Figure 1.8.2 Table of values of optimal reward-to-go, obtained by exact DP, and corresponding optimal policy [cf. the algorithm (1.96)-(1.99). Only the states x_k that are reachable from x_0 at time k are considered (this is the state space for time k).

Then the bounds $0 \leq x_k$ and $x_k \leq \bar{x}$ never become "active," and it can be verified that the optimal expected reward is $J^*(x_0) = x_0$, while all policies are optimal and attain this optimal expected reward.

(b) Suppose the investor adopts a heuristic, referred to as base heuristic, whereby he/she sells the stock if its price is greater or equal to βx_0, where β is some number with $\beta > 1$. Write an exact DP algorithm to compute the expected value of the sale under this heuristic.
Solution: The reward-to-go for the base heuristic starting from state x_k, denoted $J_k^{x_k}(x_k)$, can be generated by the following (exact) DP algorithm. (Note here the use of superscript x_k in the quantities $J_n^{x_k}(x_n)$ computed by the algorithm. The reason is that the computed values $J_n^{x_k}(x_n)$ depend on x_k, which incidentally implies that base heuristic is not sequentially consistent, as defined later in Section 2.3.2 of this book.) The algorithm is given by

$$J_N^{x_k}(x_N) = x_N, \tag{1.100}$$

and for $n = k, \ldots, N-1$, if $0 < x_n < \beta x_k$, then

$$J_n^{x_k}(x_n) = p^+ J_{n+1}^{x_k}(x_n + 1) + (1 - p^+ - p^-) J_{n+1}^{x_k}(x_n) + p^- J_{n+1}^{x_k}(x_n - 1), \tag{1.101}$$

if $x_n = 0$, then

$$J_n^{x_k}(0) = p^+ J_{n+1}^{x_k}(1) + (1 - p^+) J_{n+1}^{x_k}(0), \tag{1.102}$$

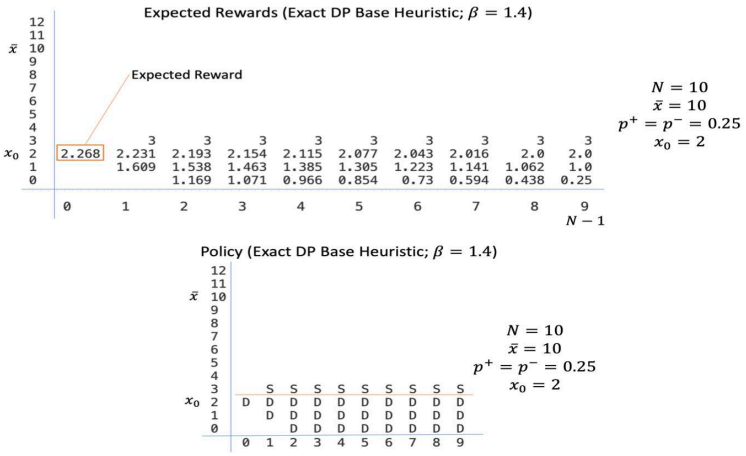

Figure 1.8.3 Table of rewards-to-go for the base policy with $\beta = 1.4$, starting from x_0 [cf. the algorithm (1.100)-(1.103) for $k = 0$].

and if $x_n \geq \beta x_k$, then

$$J_n^{x_k}(x_n) = x_n. \tag{1.103}$$

The values of $J_k^{x_k}(x_k)$ computed by this algorithm are shown in Fig. 1.8.3, together with the decisions applied by the base heuristic.

While the reward-to-go for the base heuristic starting from state x_k is very simple to compute for our problem, in order to apply the rollout algorithm only the values $J_{k+1}^{x_k+1}(x_k+1)$, $J_{k+1}^{x_k}(x_k)$, and $J_{k+1}^{x_k-1}(x_k-1)$ need to be calculated for each state x_k encountered during on-line operation. Moreover, the base heuristic's reward-to-go $J_k^{x_k}(x_k)$ can also be computed on-line by Monte Carlo simulation for the relevant states x_k. This would be the principal option in a more complicated problem where the exact DP algorithm is too time-consuming.

(c) Apply approximation in value space with one-step lookahead minimization and with function approximation that is based on the heuristic of part (b). In particular, use $\tilde{J}_N(x_N) = x_N$, and for $k = 1, \ldots, N-1$, use $\tilde{J}_k(x_k)$ that is equal to the expected value of the sale when starting at x_k and using the heuristic that sells the stock when its price exceeds βx_k. Use exact DP as well as Monte Carlo simulation to compute/approximate on-line the needed values $\tilde{J}_k(x_k)$. Compare the expected values of sale price computed with the optimal, heuristic, and approximation in value space methods.

Solution: The rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ is determined by the base heuristic, where for every possible state x_k, and stage $k = 0, \ldots, N-1$, the rollout decision $\tilde{\mu}_k(x_k)$ is

$$\tilde{\mu}_k(x_k) = \text{sell at } x_k,$$

if

$$p^+ J_{k+1}^{x_k+1}(x_k+1) + (1 - p^+ - p^-) J_{k+1}^{x_k}(x_k) + p^- J_{k+1}^{x_k-1}(x_k-1) \leq x_k,$$

and

$$\tilde{\mu}_k(x_k) = \text{don't sell at } x_k,$$

otherwise. The sell or don't sell decision of the rollout algorithm is made on-line according to the preceding criterion, at each state x_k encountered during on-line operation.

Figure 1.8.4 shows the rollout policy, which is computed by the preceding equations using the rewards-to-go of the base heuristic $J_k^{x_k}(x_k)$, as given in Fig. 1.8.3. Once the rollout policy is computed, the corresponding reward function $\tilde{J}_k(x_k)$ can be calculated similar to the case of the base heuristic. Of course, during on-line operation, the rollout decision need only be computed for the states x_k encountered on-line.

The important observation when comparing Figs. 1.8.3 and 1.8.4 is that the rewards-to-go of the rollout policy are greater or equal to the ones for the base heuristic. In particular, starting from x_0, the rollout policy attains reward 2.269, and the base heuristic attains reward 2.268. The optimal policy attains reward 2.4. The rollout policy reward is slightly closer to the optimal than the base heuristic reward.

The rollout reward-to-go values shown in Fig. 1.8.4 are "exact," and correspond to the favorable case where the heuristic rewards needed at x_k, $J_{k+1}^{x_k+1}(x_k+1)$, $J_{k+1}^{x_k}(x_k)$, and $J_{k+1}^{x_k-1}(x_k-1)$, are computed exactly by DP or by infinite-sample Monte Carlo simulation.

When finite-sample Monte Carlo simulation is used to approximate the needed base heuristic rewards at state x_k, i.e., $J_{k+1}^{x_k+1}(x_k+1)$, $J_{k+1}^{x_k}(x_k)$, and $J_{k+1}^{x_k-1}(x_k-1)$, the performance of the rollout algorithm will be degraded. In particular, by using a computer program to implement rollout with Monte Carlo simulation, it can be shown that when $J_{k+1}^{x_k+1}(x_k+1)$, $J_{k+1}^{x_k}(x_k)$, and $J_{k+1}^{x_k-1}(x_k-1)$ are approximated using a 20-sample Monte-Carlo simulation per reward value, the rollout algorithm achieves reward 2.264 starting from x_0. This reward is evaluated by (almost exact) 400-sample Monte Carlo simulation of the rollout algorithm.

When $J_{k+1}^{x_k+1}(x_k+1)$, $J_{k+1}^{x_k}(x_k)$, and $J_{k+1}^{x_k-1}(x_k-1)$ were approximated using a 200-sample Monte-Carlo simulation per reward value, the rollout algorithm achieves reward 2.273 [as evaluated by (almost exact) 400-sample Monte Carlo simulation of the rollout algorithm]. Thus with 20-sample simulation, the rollout algorithm performs worse than the base heuristic starting from x_0. With the more accurate 200-sample simulation, the rollout algorithm performs better than the base heuristic starting from x_0, and performs nearly as well as the optimal policy (but still somewhat worse than in the case where exact values of the needed base heuristic rewards are used (based on an "infinite" number Monte Carlo samples).

It is worth noting here that the heuristic is not a legitimate policy because at any state x_n is makes a decision that depends on the state x_k where it started. Thus the heuristic's decision at x_n depends not just on x_n, but also on the starting state x_k. However, the rollout algorithm is always an approximation in value space scheme with approximation reward $\tilde{J}_k(x_k)$ defined by the heuristic, and it provides a legitimate policy.

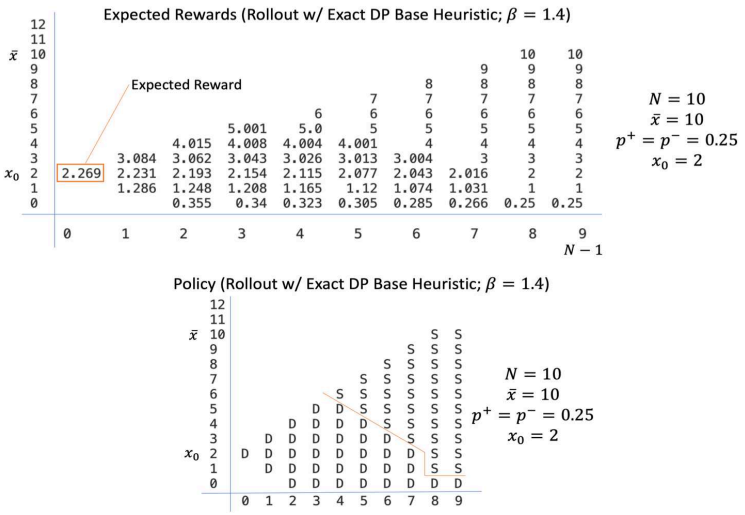

Figure 1.8.4 Table of values of reward-to-go and decisions applied by the rollout policy that corresponds to the base heuristic with $\beta = 1.4$.

(d) Repeat part (c) but with two-step instead of one-step lookahead minimization.

Answer: The implementation is very similar to the one-step lookahead case. The main difference is that at state x_k, the rollout algorithm needs to calculate the base heuristic reward values $J_{k+2}^{x_k+2}(x_k+2)$, $J_{k+2}^{x_k+1}(x_k+1)$, $J_{k+2}^{x_k}(x_k)$, $J_{k+2}^{x_k-1}(x_k-1)$, and $J_{k+2}^{x_k-2}(x_k-2)$. Thus the on-line Monte Carlo simulation work is accordingly increased. Generally the simulation work per stage of the rollout algorithm is proportional to $2\ell+1$, when ℓ-stage lookahead minimization is used, since the number of leafs at the end of the lookahead tree is $2\ell+1$.

1.5 (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.

Design an experiment to compare the performance of a fixed linear policy π, derived for a fixed nominal velocity profile, 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.

1.6 (Computational Exercise - Parking Problem)

In reference to Example 1.6.4, 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 space i being free is given, and the driver can only observe the free/taken status of a parking only after he/she visits the space. Moreover, the free/taken status of a parking 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 run-time as you increase the number of parking spots L.

1.7 (Newton's Method for Solving the Riccati Equation)

The classical form of Newton's method applied to a scalar equation of the form $H(K) = 0$ takes the form

$$K_{k+1} = K_k - \left(\frac{\partial H(K_k)}{\partial K} \right)^{-1} H(K_k), \tag{1.104}$$

where $\frac{\partial H(K_k)}{\partial K}$ is the derivative of H, evaluated at the current iterate K_k. This exercise shows algebraically (rather than graphically), within the context of linear quadratic problems, that in approximation in value space with quadratic cost approximation, the cost function of the corresponding one-step lookahead policy is the result of a Newton step for solving the Riccati equation. To this end, we will apply Newton's method to the solution of the Riccati Eq. (1.42), which we write in the form $H(K) = 0$, where

$$H(K) = K - \frac{a^2 r K}{r + b^2 K} - q. \tag{1.105}$$

(a) Show that the operation that generates K_L starting from K is a Newton iteration of the form (1.104). In other words, show that for all K that lead to a stable one-step lookahead policy, we have

$$K_L = K - \left(\frac{\partial H(K)}{\partial K}\right)^{-1} H(K), \tag{1.106}$$

where we denote by

$$K_L = \frac{q + rL^2}{1 - (a + bL)^2} \tag{1.107}$$

the quadratic cost coefficient of the one-step lookahead linear policy $\mu(x) = Lx$ corresponding to the cost function approximation $J(x) = Kx^2$:

$$L = -\frac{abK}{r + b^2K}. \tag{1.108}$$

Proof: Our approach for showing the Newton step formula (1.106) is to express each term in this formula in terms of L, and then show that the formula holds as an identity for all L. To this end, we first note from Eq. (1.108) that K can be expressed in terms of L as

$$K = -\frac{rL}{b(a + bL)}. \tag{1.109}$$

Furthermore, by using Eqs. (1.108) and (1.109), $H(K)$ as given in Eq. (1.105) can be expressed in terms of L as follows:

$$H(K) = -\frac{rL}{b(a + bL)} + \frac{arL}{b} - q. \tag{1.110}$$

Moreover, by differentiating the function H of Eq. (1.105), we obtain after a straightforward calculation

$$\frac{\partial H(K)}{\partial K} = 1 - \frac{a^2r^2}{(r + b^2K)^2} = 1 - (a + bL)^2, \tag{1.111}$$

where the second equation follows from Eq. (1.108). Having expressed all the terms in the Newton step formula (1.106) in terms of L through Eqs. (1.107), (1.109), (1.110), and (1.111), we can write this formula in terms of L only as

$$\frac{q + rL^2}{1 - (a + bL)^2} = -\frac{rL}{b(a + bL)} - \frac{1}{1 - (a + bL)^2}\left(-\frac{rL}{b(a + bL)} + \frac{arL}{b} - q\right),$$

or equivalently as

$$q + rL^2 = -\frac{rL\big(1 - (a + bL)^2\big)}{b(a + bL)} + \frac{rL}{b(a + bL)} - \frac{arL}{b} + q.$$

A straightforward calculation now shows that this equation holds as an identity for all L.

(b) What happens when K lies outside the region of stability?

(c) Show that in the case of ℓ-step lookahead, the analog of the quadratic convergence rate estimate has the form

$$|K_{\tilde{L}} - K^*| \leq c \left| F^{\ell-1}(\tilde{K}) - K^* \right|^2,$$

where $F^{\ell-1}(\tilde{K})$ is the result of the $(\ell-1)$-fold application of the mapping F to $\tilde{K}$. Thus a stronger bound for $|K_{\tilde{L}} - K^*|$ is obtained.

1.8 (Region of Stability and the Role of Multistep Lookahead)

In Section 1.5, we discussed the concept of the region of stability in the context of linear quadratic problems. The concept extends to far more general infinite horizon problems (see e.g., the book [Ber22a], Section 3.3). The idea is to call a stationary policy μ unstable if $J_\mu(x) = \infty$ for some states x, and call it stable otherwise. For $\ell \geq 1$, the ℓ-step region of stability is the set of $\tilde{J}$ for which the corresponding ℓ-step lookahead policy is stable.

Generally, the ℓ-step region of stability expands as ℓ increases. Note also that in finite-state discounted problems all policies are stable, so all $\tilde{J}$ belong to the region of stability. However, for SSP this is not so: there are policies, called *improper*, that do not terminate with positive probability for some initial states (see the books [Ber12] and [Ber22b] for extensive discussions). Such policies can be unstable. In the following example the region of instability includes functions that are very close to J^*, even with large ℓ. This example involves small stage costs, a class of problems that pose challenges for approximation in value space; see Section 2.6.

Consider a shortest path problem with a single state 1, plus the termination state t. At state 1 we can either stay at that state at cost $\epsilon > 0$ or move to the state t at cost 1. Thus the optimal policy at state 1 is to move to t, the optimal cost $J^*(1) = 1$, and is the unique solution of Bellman's equation

$$J^*(1) = \min\left\{1,\, \epsilon + J^*(1)\right\}.$$

(In SSP the optimal cost at t is 0 by assumption, and Bellman's equation involves only the costs of the states other than t.)

(a) Show that the one-step region of stability is the set of all $\tilde{J}(1) > 1 - \epsilon$. What happens in the case where $\tilde{J}(1) = 1 - \epsilon$? Show also that the ℓ-step region of stability is the set of all $\tilde{J}(1) > 1 - \ell\epsilon$. *Note*: The ℓ-step region of stability becomes arbitrarily large for sufficiently large ℓ. However, the boundary of the ℓ-step region of stability is arbitrarily close to $J^*(1)$ for sufficiently small ϵ.

(b) What happens in the case where there are additional states $i = 2, \ldots, n$, and for each of these states i there is the option of staying at i at cost ϵ or moving to $i-1$ at cost 0? *Partial answer*: The one-step region of stability consists of all $\tilde{J} = \big(\tilde{J}(n), \ldots, \tilde{J}(1)\big)$ such that $\epsilon + \tilde{J}(i) > \tilde{J}(i-1)$ for all $i \geq 2$ and $\epsilon + \tilde{J}(1) > 1$.

References

[ABB19] Agrawal, A., Barratt, S., Boyd, S., and Stellato, B., 2019. "Learning Convex Optimization Control Policies," 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:1901.08584.

[AHZ19] Arcari, E., Hewing, L., and Zeilinger, M. N., 2019. "An Approximate Dynamic Programming Approach for Dual Stochastic Model Predictive Control," arXiv:1911.03728.

[AKFJ95] Abou-Kandil, H., Freiling, Gerhard, and Jank, G., 1995. "On the Solution of Discrete-Time Markovian Jump Linear Quadratic Control Problems," Automatica, Vol. 31, pp. 765768.

[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.

[AMS19] Agostinelli, F., McAleer, S., Shmakov, A., and Baldi, P., 2019. Solving the Rubik's Cube with Deep Reinforcement Learning and Search," Nature Machine Intelligence, Vol. 1, pp. 356-363.

[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.

[ASS68] Aleksandrov, V. M., Sysoyev, V. I, and Shemeneva, V. V., 1968. "Stochastic Optimization of Systems," Engineering Cybernetics, Vol. 5, pp. 11-16.

[AXG16] Ames, A. D., Xu, X., Grizzle, J. W., and Tabuada, P., 2016. "Control Barrier Function Based Quadratic Programs for Safety Critical Systems," IEEE Transactions 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.

[Ala22] Alamir, M., 2022. "Learning Against Uncertainty in Control Engineering," Annual Reviews in Control.

[AnH14] Antunes, D., and Heemels, W.P.M.H., 2014. "Rollout Event-Triggered Control: Beyond Periodic Control Performance," IEEE Transactions 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.

[AsH06] Aström, K. J., and Hagglund, T., 2006. Advanced PID Control, Instrument Society of America, Research Triangle Park, N. C.

[AsW94] Aström, K. J., and Wittenmark, B., 1994. Adaptive Control, 2nd Edition, Prentice-Hall, Englewood Cliffs, N. J.

[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:2004.14427.

[BBB22] Bhambri, S., Bhattacharjee, A., and Bertsekas, D. P., 2022. "Reinforcement Learning Methods for Wordle: A POMDP/Adaptive Control Approach," arXiv:2211.10298.

[BBB23] Bhambri, S., Bhattacharjee, A., and Bertsekas, D. P., 2023. "Playing Wordle Using an Online Rollout Algorithm for Deterministic POMDPs," 2023 IEEE Conference on Games, Boston, MA.

[BBD08] Busoniu, L., Babuska, R., and De Schutter, B., 2008. "A Comprehensive Survey of Multiagent Reinforcement Learning," IEEE Transactions 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.

[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.

[BBS87] Bean, J. C., Birge, J. R., and Smith, R. L., 1987. "Aggregation in Dynamic Programming," Operations Research, Vol. 35, pp. 215-220.

[BBW20] Bhattacharya, S., Badyal, S., Wheeler, T., Gil, S., Bertsekas, D. P., 2020. "Reinforcement Learning for POMDP: Partitioned Rollout and Policy Iteration with Application to Autonomous Sequential Repair Problems," to appear in IEEE Robotics and Automation Letters, 2020; arXiv:2002.04175.

[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: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.

[BFA22] Bouguila, N., Fan, W. and Amayri, M., eds., 2022. Hidden Markov Models and Applications. Springer, NY.

[BFH86] Breton, M., Filar, J. A., Haurie, A., and Schultz, T. A., 1986. "On the Computation of Equilibria in Discounted Stochastic Dynamic Games," in Dynamic Games and Applications in Economics, Springer, pp. 64-87.

[BLJ23] Bai, T., Li, Y., Johansson, K. H., and Martensson, J., 2023. "Rollout-Based Charging Strategy for Electric Trucks with Hours-of-Service Regulations," arXiv Preprint, arXiv:2303.08895.

[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.

[BKM05] de Boer, P. T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. 2005. "A Tutorial on the Cross-Entropy Method," Annals of Operations Research, Vol. 134, pp. 19-67.

[BLL19] Bartlett, P. L., Long, P. M., Lugosi, G., and Tsigler, A., 2019. "Benign Overfitting in Linear Regression," arXiv:1906.11300.

[BMM18] Belkin, M., Ma, S., and Mandal, S., 2018. "To Understand Deep Learning we Need to Understand Kernel Learning," arXiv:1802.01396.

[BMM24] Bhusal, G., Miller, K., and Merkurjev, E., 2024. "MALADY: Multiclass Active Learning with Auction Dynamics on Graphs," arXiv preprint arXiv:2409.09475.

[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:1806.09471.

[BSA83] Barto, A. G., Sutton, R. S., and Anderson, C. W., 1983. "Neuronlike Elements that Can Solve Difficult Learning Control Problems," IEEE Trans. on Systems, Man, and Cybernetics, Vol. 13, pp. 835-846.

[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: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.

[BaB01] Baxter, J., and Bartlett, P. L., 2001. "Infinite-Horizon Policy-Gradient Estimation," Journal of Artificial Intelligence Research, Vol. 15, pp. 319-350.

[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.

[BaP12] Bauso, D., and Pesenti, R., 2012. "Team Theory and Person-by-Person Optimization with Binary Decisions," SIAM Journal on Control and Optimization, Vol. 50, pp. 3011-3028.

[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.

[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.

[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.

[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 in 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 Edition, 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, MIT.

[Bel56] Bellman, R., 1956. "A Problem in the Sequential Design of Experiments," Sankhya: The Indian Journal of Statistics, Vol. 16, pp. 221-229.

[Bel57] Bellman, R., 1957. Dynamic Programming, Princeton University Press, Princeton, N. J.

[Bel84] Bellman, R., 1984. Eye of the Hurricane, World Scientific Publishing, Singapore.

[Bel87] Bellman, R., 1987. Introduction to the Mathematical Theory of Control Processes, Academic Press, Vols. I and II, New York, N. Y.

[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. Dissertation, Massachusetts Institute of Technology, Cambridge, MA (can be downloaded from the author's website).

[Ber72a] 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.

[Ber72b] Bertsekas, D. P., 1972. "On the Solution of Some Minimax Control Problems," Proc. 1972 IEEE Decision and Control Conf., New Orleans, LA.

[Ber73] Bertsekas, D. P., 1973. "Linear Convex Stochastic Control Problems over an Infinite Horizon," IEEE Trans. Automatic Control, Vol. AC-18, pp. 314-315.

[Ber76] Bertsekas, D. P., 1976. Dynamic Programming and Stoschastic Control, Academic Press, NY (can be downloaded fro the author's website).

[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, MIT, 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, MIT.

[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, MIT; 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.

[Ber10c] Bertsekas, D. P., 2010. "Pathologies of Temporal Difference Methods in Approximate Dynamic Programming," Proc. 2010 IEEE Conference on Decision and Control, Atlanta, GA, Dec. 2010.

[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.

[Ber11c] Bertsekas, D. P., 2011. "Temporal Difference Methods for General Projected Equations," IEEE Trans. on Automatic Control, Vol. 56, pp. 2128-2139.

[Ber12] Bertsekas, D. P., 2012. Dynamic Programming and Optimal Control, Vol. II, 4th Edition, 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, MIT; arXiv:1507.1365936.

[Ber16] Bertsekas, D. P., 2016. Nonlinear Programming, 3rd Edition, Athena Scientific, Belmont, MA.

[Ber17a] Bertsekas, D. P., 2017. Dynamic Programming and Optimal Control, Vol. I, 4th Edition, Athena Scientific, Belmont, MA.

[Ber17b] Bertsekas, D. P., 2017. "Value and Policy Iteration in Deterministic Optimal Control and Adaptive Dynamic Programming," IEEE Transactions on Neural Networks and Learning Systems, Vol. 28, pp. 500-509.

[Ber18a] Bertsekas, D. P., 2018. "Feature-Based Aggregation and Deep Reinforcement Learning: A Survey and Some New Implementations," Lab. for Information and Decision Systems Report, MIT; arXiv:1804.04577; IEEE/CAA Journal of Automatica Sinica, Vol. 6, 2019, pp. 1-31.

[Ber18b] Bertsekas, D. P., 2018. "Biased Aggregation, Rollout, and Enhanced Policy Improvement for Reinforcement Learning," Lab. for Information and Decision Systems Report, MIT; arXiv:1910.02426.

[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:1910.00120.

[Ber19d] Bertsekas, D. P., 2019. "Constrained Multiagent Rollout and Multidimensional Assignment with the Auction Algorithm," arXiv preprint, arxiv:2002.07407.

[Ber20a] Bertsekas, D. P., 2020. Rollout, Policy Iteration, and Distributed Reinforcement Learning, Athena Scientific, Belmont, MA.

[Ber20b] Bertsekas, D. P., 2020. "Multiagent Value Iteration Algorithms in Dynamic Programming and Reinforcement Learning," arXiv preprint, arxiv.org/abs/2005.01627; appears in Results in Control and Optimization Journal, Vol. 1, 2020.

[Ber21a] Bertsekas, D. P., 2021. "Multiagent Reinforcement Learning: Rollout and Policy Iteration," IEEE/CAA Journal of Automatica Sinica, Vol. 8, pp. 249-271.

[Ber21b] Bertsekas, D. P., 2021. "Distributed Asynchronous Policy Iteration for Sequential Zero-Sum Games and Minimax Control," arXiv:2107.10406

[Ber22a] Bertsekas, D. P., 2022. Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control, Athena Scientific, Belmont, MA.

[Ber22b] Bertsekas, D. P., 2022. Abstract Dynamic Programming, 3rd Edition, Athena Scientific, Belmont, MA (can be downloaded from the author's website).

[Ber22c] Bertsekas, D. P., 2022. "Newton's Method for Reinforcement Learning and Model Predictive Control," Results in Control and Optimization, Vol. 7, pp. 100-121.

[Ber22d] Bertsekas, D. P., 2022. "Rollout Algorithms and Approximate Dynamic Programming for Bayesian Optimization and Sequential Estimation," arXiv:2212.07998.

[Ber24] Bertsekas, D. P., 2024. "Model Predictive Control, and Reinforcement Learning: A Unified Framework Based on Dynamic Programming," arXiv preprint arXiv:2406.00592; to be published in Proc. IFAC NMPC.

[Bet10] Bethke, B. M., 2010. Kernel-Based Approximate Dynamic Programming Using Bellman Residual Elimination, Ph.D. Thesis, MIT.

[BiB24] Bishop, C. M, and Bishop, H., 2024. Deep Learning: Foundations and Concepts, Springer, New York, N. Y.

[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.

[BoV79] Borkar, V. and Varaiya, P., 1979. "Adaptive Control of Markov Chains, I: Finite Parameter Set," IEEE Trans. on Automatic Control, Vol. 24, pp. 953-957.

[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 Transactions on Robotics, Vol. 25, pp. 912-926.

[CFH05] 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.

[CFH13] Chang, H. S., Hu, J., Fu, M. C., and Marcus, S. I., 2013. Simulation-Based Algorithms for Markov Decision Processes, 2nd Edition, Springer, N. Y.

[CFM05] Costa, O. L. V., Fragoso, M. D., and Marques, R. P., 2005. Discrete-Time Markov Jump Linear Sys- tems, Springer Science and Business Media.

[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: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.

[CWC86] Chizeck, H. J., Willsky, A. S., and Castanon, D., "Discrete- Time Markovian-Jump Linear Quadratic Optimal Control," International Journal of Control, Vol. 43, pp. 213231.

[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 Edition, Springer, New York, N. Y.

[CaC97] Cao, X. R., and Chen, H. F., 1997. "Perturbation Realization Potentials and Sensitivity Analysis of Markov Processes," IEEE Trans. on Aut. Control, Vol. 32, pp. 1382-1393.

[CaW98] Cao, X. R., and Wan, Y. W., 1998. "Algorithms for Sensitivity Analysis of Markov Systems Through Potentials and Perturbation Realization," IEEE Trans. Control Systems Technology, Vol. 6, pp. 482-494.

[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.

[Cha24] Chang, H. S., 2024. "On the Convergence Rate of MCTS for the Optimal Value Estimation in Markov Decision Processes," arXiv preprint arXiv:2402.07063.

[Che72] Chernoff, H., 1972. "Sequential Analysis and Optimal Design," Regional Conference Series in Applied Mathematics, SIAM, Philadelphia, PA.

[ChM82] Chatelin, F., and Miranker, W. L., 1982. "Acceleration by Aggregation of Successive Approximation Methods," Linear Algebra and its Applications, Vol. 43, pp. 17-47.

[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.

[CiS15] Ciosek, K., and Silver, D., 2015. "Value Iteration with Options and State Aggregation," arXiv preprint arXiv:1501.03959.

[Cio15] Ciosek, K. A., 2015. Linear Reinforcement Learning with Options, Doctoral Dissertation, University College London.

[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:1905.02199.

[CDV02] Costa, E. F., and Do Val, J. B. R., 2002. "Weak Detectability and the Linear-Quadratic Control Problem of Discrete-Time Markov Jump Linear Systems," International J. of Control, Vol. 75, pp. 1282 1292.

[DEK98] Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G., 1998. Biological Sequence Analysis, Cambridge Univ. Press, Cambridge.

[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.

[DHS11] Duchi, J., Hazan, E., and Singer, Y., 2011. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," J. of Machine Learning Research, Vol. 12, pp. 2121-2159.

[DHS12] Duda, R. O., Hart, P. E., and Stork, D. G., 2012. Pattern Classification, J. Wiley, N. Y.

[DJW12] Duchi, J., Jordan, M. I., Wainwright, M. J., and Wibisono, A., 2012. "Finite Sample Convergence Rate of Zero-Order Stochastic Optimization Methods," NIPS, pp. 1448-1456.

[DJW15] Duchi, J., Jordan, M. I., Wainwright, M. J., and Wibisono, A., 2015. "Optimal Rates for Zero-Order Convex Optimization: The Power of Two Function Evaluations," IEEE Trans. on Information Theory, Vol. 61, pp. 2788-2806.

[DNP11] Deisenroth, M. P., Neumann, G., and Peters, J., 2011. "A Survey on Policy Search for Robotics," Foundations and Trends in Robotics, Vol. 2, pp. 1-142.

[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.

[DeF04] De Farias, D. P., 2004. "The Linear Programming Approach to Approximate Dynamic Programming," in Learning and Approximate Dynamic Programming, by J. Si, A. Barto, W. Powell, and D. Wunsch, (Eds.), IEEE Press, N. Y.

[DeG70] DeGroot, M. H., 1970. Optimal Statistical Decisions, McGraw-Hill, 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.

[DoD93] Douglas, C. C., and Douglas, J., 1993. "A Unified Convergence Theory for Abstract Multigrid or Multilevel Algorithms, Serial and Parallel," SIAM J. Num. Anal., Vol. 30, pp. 136-158.

[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.

[Dre65] Dreyfus, S. D., 1965. Dynamic Programming and the Calculus of Variations, Academic Press, N. Y.

[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.

[ELP12] Estanjini, R. M., Li, K., and Paschalidis, I. C., 2012. "A Least Squares Temporal Difference Actor-Critic Algorithm with Applications to Warehouse Management," Naval Research Logistics, Vol. 59, pp. 197-211.

[EMM05] Engel, Y., Mannor, S., and Meir, R., 2005. "Reinforcement Learning with Gaussian Processes," in Proc. of the 22nd ICML, pp. 201-208.

[EPE20] Emami, P., Pardalos, P. M., Elefteriadou, L., and Ranka, S., 2020. "Machine Learning Methods for Data Association in Multi-Object Tracking," ACM Computing Surveys (CSUR), Vol. 53, pp. 1-34.

[Edd96] Eddy, S. R., 1996. "Hidden Markov Models," Current Opinion in Structural Biology, Vol. 6, pp. 361-365.

[EpM02] Ephraim, Y., and Merhav, N., 2002. "Hidden Markov Processes," IEEE Trans. on Information Theory, Vol. 48, pp. 1518-1569.

[FHS09] Feitzinger, F., Hylla, T., and Sachs, E. W., 2009. "Inexact Kleinman-Newton Method for Riccati Equations," SIAM Journal 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 Journal 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.

[Fel63] Feldbaum, A. A., 1963. "Dual Control Theory Problems," IFAC Proceedings, pp. 541-550.

[FiT91] Filar, J. A., and Tolwinski, B., 1991. "On the Algorithm of Pollatschek and Avi-ltzhak," in Stochastic Games and Related Topics, Theory and Decision Library, Springer, Vol. 7, pp. 59-70.

[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.

[For73] Forney, G. D., 1973. "The Viterbi Algorithm," Proc. IEEE, Vol. 61, pp. 268-278.

[Fra18] Frazier, P. I., 2018. "A Tutorial on Bayesian Optimization," 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.

[FuH94] Fu, M. C., and Hu, J.-Q., 1994. "Smoothed Perturbation Analysis Derivative Estimation for Markov Chains," Oper. Res. Letters, Vol. 41, pp. 241-251.

[Fun89] Funahashi, K., 1989. "On the Approximate Realization of Continuous Mappings by Neural Networks," Neural Networks, Vol. 2, pp. 183-192.

[GBB04] Greensmith, E., Bartlett, P. L., and Baxter, J., 2004. "Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning," Journal of Machine Learning Research, Vol. 5, pp. 1471-1530.

[GBC16] Goodfellow, I., Bengio, J., and Courville, A., Deep Learning, MIT Press, Cambridge, MA.

[GBL12] Grondman, I., Busoniu, L., Lopes, G. A. D., and Babuska, R., 2012. "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients," IEEE Trans. on Systems, Man, and Cybernetics, Part C, Vol. 42, pp. 1291-1307.

[GFB24] Gioia, D. G., Fadda, E., and Brandimarte, P., 2024. "Rolling Horizon Policies for Multi-Stage Stochastic Assemble-to-Order Problems," International Journal of Production Research, Vol. 62, pp. 5108-5126.

[GLB24] Gundawar, A., Li, Y., and Bertsekas, D., 2024. "Playing Superior Computer Chess with Model Predictive Control and Rollout," arXiv preprint, arXiv:2409.06477.

[GPG22] Garces, D., Bhattacharya, S., Gil, G., and Bertsekas, D., "Multiagent Reinforcement Learning for Autonomous Routing and Pickup Problem with Adaptation to Variable Demand," arXiv preprint arXiv:2211.14983.

[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.

[GHC21] Gerlach, T., Hoffmann, F., and Charlish, A., 2021. "Policy Rollout Action Selection with Knowledge Gradient for Sensor Path Planning," 2021 IEEE 24th International Conference on Information Fusion, pp. 1-8.

[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 Journal of Operational Research, Vol. 258, pp. 216-229.

[GeB13] Geffner, H., and Bonet, B., 2013. A Concise Introduction to Models and Methods for Automated Planning, Morgan and Claypool Publishers.

[GeP24] Gerlach, T., and Piatkowski, N., 2024. "Dynamic Range Reduction via Branch-and-Bound," arXiv preprint arXiv:2409.10863.

[Gly87] Glynn, P. W., 1987. "Likelihood Ratio Gradient Estimation: An Overview," Proc. of the 1987 Winter Simulation Conference, pp. 366-375.

[Gly90] Glynn, P. W., 1990. "Likelihood Ratio Gradient Estimation for Stochastic Systems," Communications of the ACM, Vol. 33, pp. 75-84.

[GoS84] Goodwin, G. C., and Sin, K. S. S., 1984. Adaptive Filtering, Prediction, and Control, Prentice-Hall, Englewood Cliffs, N. J.

[Gor95] Gordon, G. J., 1995. "Stable Function Approximation in Dynamic Programming," in Machine Learning: Proceedings of the Twelfth International Conference, Morgan Kaufmann, San Francisco, CA.

[Gos15] Gosavi, A., 2015. Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning, 2nd Edition, 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.

[HLS06] Han, J., Lai, T. L. and Spivakovsky, V., 2006. "Approximate Policy Optimization and Adaptive Control in Regression Models," Computational Economics, Vol. 27, pp. 433-452.

[HMR19] Hastie, T., Montanari, A., Rosset, S., and Tibshirani, R. J., 2019. "Surprises in High-Dimensional Ridgeless Least Squares Interpolation," arXiv:1903.08560.

[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.

[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.

[HWP22] Hu, J., Wang, Y., Pang, Y., and Liu, Y., 2022. "Optimal Maintenance Scheduling under Uncertainties using Linear Programming-Enhanced Reinforcement Learning," Engineering Applications of Artificial Intelligence, Vol. 109.

[HaR21] Hardt, M., and Recht, B., 2021. Patterns, Predictions, and Actions: A Story About Machine Learning, arXiv:2102.05242; published by Priceton Univ. Press, 2022.

[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 Edition, 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.

[HuM16] Huan, X., and Marzouk, Y. M., 2016. "Sequential Bayesian Optimal Experimental Design via Approximate Dynamic Programming," arXiv:1604.08320.

[Hua15] Huan, X., 2015. Numerical Approaches for Sequential Bayesian Optimal Experimental Design, Ph.D. Thesis, MIT.

[Hyl11] Hylla, T., 2011. Extension of Inexact Kleinman-Newton Methods to a General Monotonicity Preserving Convergence Theory, PhD Thesis, Univ. of Trier.

[IFT19] Issakkimuthu, M., Fern, A., and Tadepalli, P., 2019. "The Choice Function Framework for Online Policy Improvement," arXiv:1910.00614.

[IJT18] Iusem, A., 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 Journal 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.

[JSJ95] Jaakkola, T., Singh, S. P., and Jordan, M. I., 1995. "Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems," NIPS, Vol. 7, pp. 345-352.

[JSW98] Jones, D. R., Schonlau, M., and Welch, W. J., 1998. "Efficient Global Optimization of Expensive Black-Box Functions," J. of Global Optimization, Vol. 13, pp. 455?492.

[JiJ17] Jiang, Y., and Jiang, Z. P., 2017. Robust Adaptive Dynamic Programming, J. Wiley, N. Y.

[JoB16] Joseph, A. G., and Bhatnagar, S., 2016. "Revisiting the Cross Entropy Method with Applications in Stochastic Global Optimization and Reinforcement Learning," in Proc. of the 22nd European Conference on Artificial Intelligence, pp. 1026-1034.

[JoB18] Joseph, A. G., and Bhatnagar, S., 2018. "A Cross Entropy Based Optimization Algorithm with Global Convergence Guarantees," arXiv preprint arXiv:1801.10291.

[Jon90] Jones, L. K., 1990. "Constructive Approximations for Neural Networks by Sigmoidal Functions," Proceedings of the IEEE, Vol. 78, pp. 1586-1589.

[JuM23] Jurafsky, D., and Martin, J. H., 2023. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, draft 3rd edition (on-line).

[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.

[KGB82] Kimemia, J., Gershwin, S. B., and Bertsekas, D. P., 1982. "Computation of Production Control Policies by a Dynamic Programming Technique," in Analysis and Optimization of Systems, A. Bensoussan and J. L. Lions (eds.), Springer, N. Y., pp. 243-269.

[KKK95] Krstic, M., Kanellakopoulos, I., Kokotovic, P., 1995. Nonlinear and Adaptive Control Design, J. Wiley, N. Y.

[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 Transactions 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 Transactions on Automatic Control, Vol. 27, pp. 848-859.

[KLM96] Kaelbling, L. P., Littman, M. L., and Moore, A. W., 1996. "Reinforcement Learning: A Survey," J. of Artificial Intelligence Res., Vol. 4, pp. 237-285.

[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.

[KRC13] Kroese, D. P., Rubinstein, R. Y., Cohen, I., Porotsky, S., and Taimre, T., 2013. "Cross-Entropy Method," in Encyclopedia of Operations Research and Management Science, Springer, Boston, MA, pp. 326-333.

[KaW94] Kall, P., and Wallace, S. W., 1994. Stochastic Programming, Wiley, Chichester, UK.

[Kak02] Kakade, S. A., 2002. "Natural Policy Gradient," NIPS, Vol. 14, pp. 1531-1538.

[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.

[KiB14] Kingma, D. P., and Ba, J., 2014. "Adam: A Method for Stochastic Optimization," arXiv preprint arXiv:1412.6980.

[Kir04] Kirk, D. E., 2004. Optimal Control Theory: An Introduction, Courier Corporation.

[Kim82] Kimemia, J., 1982. "Hierarchical Control of Production in Flexible Manufacturing Systems," Ph.D. Thesis, Dep. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

[Kle09] Kleijnen, J. P., 2009. "Kriging Metamodeling in Simulation: A Review," European Journal of Operational Research, Vol. 192, pp. 707-716.

[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.

[KoT99] Konda, V. R., and Tsitsiklis, J. N., 1999. "Actor-Critic Algorithms," NIPS, Denver, Colorado, pp. 1008-1014.

[KoT03] Konda, V. R., and Tsitsiklis, J. N., 2003. "Actor-Critic Algorithms," SIAM J. on Control and Optimization, Vol. 42, pp. 1143-1166.

[Kre19] Krener, A. J., 2019. "Adaptive Horizon Model Predictive Control and Al'brekht's Method," arXiv:1904.00053.

[Kri16] Krishnamurthy, V., 2016. Partially Observed Markov Decision Processes, Cambridge Univ. Press.

[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.

[L'Ec91] L'Ecuyer, P., 1991. "An Overview of Derivative Estimation," Proceedings of the 1991 Winter Simulation Conference, pp. 207-217.

[LEC20] Lee, E. H., Eriksson, D., Cheng, B., McCourt, M., and Bindel, D., 2020. "Efficient Rollout Strategies for Bayesian Optimization," arXiv:2002.10539.

[LEP21] Lee, E. H., Eriksson, D., Perrone, V., and Seeger, M., 2021. "A Nonmyopic Approach to Cost-Constrained Bayesian Optimization," In Uncertainty in Artificial Intelligence Proceedings, pp. 568-577.

[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:1912.08504.

[LJM21] Li, Y., Johansson, K. H., Martensson, J., and Bertsekas, D. P., 2021. "Data-Driven Rollout for Deterministic Optimal Control," arXiv preprint arXiv:2105.03116.

[LKG21] Li, T., Krakow, L. W., and Gopalswamy, S., 2021. "Optimizing Consensus-Based Multi-Target Tracking with Multiagent Rollout Control Policies," In 2021 IEEE Conference on Control Technology and Applications, pp. 131-137.

[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.

[LPS22] Liu, M., Pedrielli, G., Sulc, P., Poppleton, E., Bertsekas, D. P., 2022. "ExpertRNA: A New Framework for RNA Structure Prediction," INFORMS Journal on Computing.

[LRD23] Laidlaw, C., Russell, S., and Dragan, A., 2023. "Bridging RL Theory and Practice with the Effective Horizon," arXiv preprint arXiv:2304.09853.

[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: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.

[LaS20] Lattimore, T., and Szepesvari, C., 2020. Bandit Algorithms, Cambridge University Press.

[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.

[LiB24] Li, Y., and Bertsekas, D. P., 2024. "Most Likely Sequence Generation for n-Grams, Transformers, HMMs, and Markov Chains, by Using Rollout Algorithms," arXiv:2403.15465.

[LiB25] Li, Y., and Bertsekas, D. P., 2025. "Semilinear Dynamic Programming: Analysis, Algorithms, and Certainty Equivalence Properties," arXiv:2501.04668, 2025.

[LiS16] Liang, S., and Srikant, R., 2016. "Why Deep Neural Networks for Function Approximation?" 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.

[LoC23] Loxley, P. N., and Cheung, K. W., 2023. "A Dynamic Programming Algorithm for Finding an Optimal Sequence of Informative Measurements," Entropy, Vol. 25, p. 251.

[MAS19] McAleer, S., Agostinelli, F., Shmakov, A. K. .and Baldi, P., 2019. "Solving the Rubik's Cube with Approximate Policy Iteration," in International Conference on Learning Representations.

[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.

[MDM01] Magni, L., De Nicolao, G., Magnani, L., and Scattolini, R., 2001. "A Stabilizing Model-Based Predictive Control Algorithm for Nonlinear Systems," Automatica, Vol. 37, pp. 1351-1362.

[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.

[MLW24] Musunuru, P., Li, Y., Weber, J., and Bertsekas, D., "An Approximate Dynamic Programming Framework for Occlusion-Robust Multi-Object Tracking," arXiv: 2405.15137, May 2024.

[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.

[MMK23] Marchesoni-Acland, F., Morel, J. M., Kherroubi, J., and Facciolo, G., 2023. "Optimal and Efficient Binary Questioning for Human-in-the-Loop Annotation," arXiv preprint arXiv:2307.01578.

[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.

[MRG03] Mannor, S., Rubinstein, R. Y., and Gat, Y., 2003. "The Cross Entropy Method for Fast Policy Search," in Proc. of the 20th International Conference on Machine Learning (ICML-03), pp. 512-519.

[MRR00] Mayne, D., Rawlings, J. B., Rao, C. V., and Scokaert, P. O. M., 2000. "Con-

strained Model Predictive Control: Stability and Optimality," Automatica, Vol. 36, pp. 789-814.

[MVS19] Muthukumar, V., Vodrahalli, K., and Sahai, A., 2019. "Harmless Interpolation of Noisy Data in Regression," 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.

[MaE07] Mamon, R. S., and Elliott, R. J. eds., 2007. Hidden Markov Models in Finance, Springer, NY.

[MaE14] Mamon, R. S., and Elliott, R. J. eds., 2007. Hidden Markov Models in Finance: Further Developments and Applications, Springer, NY.

[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.

[MaS99] Manning, C., and Schutze, H., 1999. Foundations of Statistical Natural Language Processing, MIT Press, Cambridge, MA.

[MaS02] Martinez, L., and Soares, S., 2002. "Comparison Between Closed-Loop and Partial Open-Loop Feedback Control Policies in Long Term Hydrothermal Scheduling," IEEE Transactions on Power Systems, Vol. 17, pp. 330-336.

[MaT01] Marbach, P., and Tsitsiklis, J. N., 2001. "Simulation-Based Optimization of Markov Reward Processes," IEEE Trans. on Aut. Control, Vol. 46, pp. 191-209.

[MaT03] Marbach, P., and Tsitsiklis, J. N., 2003. "Approximate Gradient Methods in Policy-Space Optimization of Markov Reward Processes," J. Discrete Event Dynamic Systems, Vol. 13, pp. 111-148.

[Mac02] Maciejowski, J. M., 2002. Predictive Control with Constraints, Addison-Wesley, Reading, MA.

[Mar55] Marschak, J., 1955. "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.

[Mar90] Mariton, M., 1990. Jump Linear Systems in Automatic Control, CRC Press.

[Mat65] J. Matyas, J., 1965. "Random Optimization," Automation and Remote Control, Vol. 26, pp. 246-253.

[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:2007.12933.

[Mey07] Meyn, S., 2007. Control Techniques for Complex Networks, Cambridge Univ. Press, N. Y.

[Mey22] Meyn, S., 2022. Control Systems and Reinforcement Learning, 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.

[NMT13] Nayyar, A., Mahajan, A. and Teneketzis, D., 2013. "Decentralized Stochastic Control with Partial History Sharing: A Common Information Approach," IEEE Transactions on Automatic Control, Vol. 58, pp. 1644-1658.

[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.

[NSW05] Nedich, A., Schneider, M. K., and Washburn, R. B., 2005. "Farsighted Sensor Management Strategies for Move/Stop Tracking," in Proc. of 7th International Conference on Information Fusion, Vol. 1, pp. 566-573.

[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.

[NeS17] Nesterov, Y., and Spokoiny, V., 2017. "Random Gradient-Free Minimization of Convex Functions," Foundations of Computational Mathematics, Vol. 17, pp. 527-566.

[Ned11] Nedić, A., 2011. "Random Algorithms for Convex Minimization Problems," Math. Programming, Ser. B, Vol. 129, pp. 225-253.

[NoS09] Novoa, C. and Storer, R., 2009. "An Approximate Dynamic Programming Approach for the Vehicle Routing Problem with Stochastic Demands," European J. of Operational Research, Vol. 196, pp. 509-515.

[OrS02] Ormoneit, D., and Sen, S., 2002. "Kernel-Based Reinforcement Learning," Machine Learning, Vol. 49, pp. 161-178.

[PBK08] Patek, S. D., Breton, M., and Kovatchev, B. P., 2008. "Rollout Policies for Control of Blood Glucose," XIV Latin Ibero-American Congress on Operations Research - Book of Extended Abstracts.

[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 Transactions on

Aerospace and Electronic Systems, Vol. 37, pp. 22-39.

[PSC22] Paulson, J. A., Sonouifar, F., and Chakrabarty, A., 2022. "Efficient Multi-Step Lookahead Bayesian Optimization with Local Search Constraints," IEEE Conf. on Decision and Control, pp. 123-129.

[PPG16] Perolat, J., Piot, B., Geist, M., Scherrer, B., and Pietquin, O., 2016. "Softened Approximate Policy Iteration for Markov Games," in Proc. International Conference on Machine Learning, pp. 1860-1868.

[PSP15] Perolat, J., Scherrer, B., Piot, B., and Pietquin, O., 2015. "Approximate Dynamic Programming for Two-Player Zero-Sum Markov Games," in Proc. International Conference on Machine Learning, pp. 1321-1329.

[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.

[PeS08] Peters, J., and Schaal, S., 2008. "Reinforcement Learning of Motor Skills with Policy Gradients," Neural Networks, Vol. 4, pp. 682-697.

[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 Edition, J. Wiley and Sons, Hoboken, N. J.

[PrS01] Proakis, J. G., and Salehi, M., 2001. Communication Systems Engineering, Prentice-Hall, Englewood Cliffs, N. J.

[PrS08] Proakis, J. G., and Salehi, M., 2008. Digital Communications, McGraw-Hill, N. Y.

[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.

[RDM24] Ruoss, A., Delétang, G., Medapati, S., Grau-Moya, J., Wenliang, L. K., Catt, E., Reid, J., and Genewein, T., 2024. "Grandmaster-Level Chess Without Search," arXiv:2402.04494.

[RGG21] Rimélé, A., Grangier, P., Gamache, M., Gendreau, M., and Rousseau, L. M., 2021. "E-Commerce Warehousing: Learning a Storage Policy, arXiv:2101.08828.

[RMD17] Rawlings, J. B., Mayne, D. Q., and Diehl, M. M., 2017. Model Predictive Control: Theory, Computation, and Design, 2nd Ed., Nob Hill Publishing.

[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.

[RPW91] Rogers, D. F., Plante, R. D., Wong, R. T., and Evans, J. R., 1991. "Aggregation and Disaggregation Techniques and Methodology in Optimization," Operations Research, Vol. 39, pp. 553-582.

[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.

[RST23] Rusmevichientong, P., Sumida, M., Topaloglu, H., and Bai, Y., 2023. "Revenue Management with Heterogeneous Resources: Unit Resource Capacities, Advance Bookings, and Itineraries over Time Intervals," Operations Research, Articles in Advance.

[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.

[Rab89] Rabiner, L. R., 1989. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," Proc. of the IEEE, Vol. 77, pp. 257-286.

[Rad62] Radner, R., 1962. "Team Decision Problems," Ann. Math. Statist., Vol. 33, pp. 857-881.

[Ras63] Rastrigin, R. A., 1963. "About Convergence of Random Search Method in Extremal Control of Multi-Parameter Systems," Avtomat. i Telemekh., Vol. 24, pp. 1467-1473.

[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 Edition, 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.

[RuK04] Rubinstein, R. Y., and Kroese, D. P., 2004. The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Springer, N. Y.

[RuK13] Rubinstein, R. Y., and Kroese, D. P., 2013. The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning, Springer Science and Business Media.

[RuK16] Rubinstein, R. Y., and Kroese, D. P., 2016. Simulation and the Monte Carlo Method, 3rd Edition, 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.

[Rub69] Rubinstein, R. Y., 1969. Some Problems in Monte Carlo Optimization, Ph.D. Thesis.

[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:1712.01815.

[SHS24] Samani, F. S., Hammar, K., and Stadler, R., 2024. "Online Policy Adaptation for Networked Systems using Rollout," In NOMS 2024-2024 IEEE Network Operations and Management Symposium, pp. 1-9.

[SJJ95] Singh, S. P., Jaakkola, T., and Jordan, M. I., 1995. "Reinforcement Learning with Soft State Aggregation," in Advances in Neural Information Processing Systems 7, MIT Press, Cambridge, MA.

[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.

[SLD24] Samani, F. S., Larsson, H., Damberg, S., Johnsson, A., and Stadler, R., 2024. "Comparing Transfer Learning and Rollout for Policy Adaptation in a Changing Network Environment," in 2024 IEEE Network Operations and Management Symposium.

[SMS99] Sutton, R. S., McAllester, D., Singh, S. P., and Mansour, Y., 1999. "Policy Gradient Methods for Reinforcement Learning with Function Approximation," NIPS, Denver, Colorado.

[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.

[SPS99] Sutton, R. S., Precup, D., and Singh, S., 1999. "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," Artificial Intelligence, Vol. 112, pp. 181-211.

[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.

[SWD17] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., 2017. "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347.

[SWM89] Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., 1989. "Design and Analysis of Computer Experiments," Statistical Science, pp. 409-423.

[SXX22] Shah, D., Xie, Q., and Xu, Z., 2022. "Nonasymptotic Analysis of Monte Carlo Tree Search," Operations Research, vol. 70, pp. 3234-3260.

[SYL17] Saldi, N., Yuksel, S., and Linder, T., 2017. "Finite Model Approximations for Partially Observed Markov Decision Processes with Discounted Cost," 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.

[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, PhD 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.

[SiB22] Silver, D., and Barreto, A., 2022. "Simulation-Based Search," in Proc. Int. Cong. Math, Vol. 6, pp. 4800-4819.

[SiK19] Singh, R., and Kumar, P. R., 2019. "Optimal Decentralized Dynamic Policies for Video Streaming over Wireless Channels," arXiv:1902.07418.

[SlL91] Slotine, J.-J. E., and Li, W., Applied Nonlinear Control, Prentice-Hall, Englewood Cliffs, N. J.

[Spa92] Spall, J. C., "Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation," IEEE Trans. on Automatic Control, Vol. pp. 332-341.

[Spa03] Spall, J. C., 2003. Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control, J. Wiley, Hoboken, N. J.

[StW91] Stewart, B. S., and White, C. C., 1991. "Multiobjective A^*," J. ACM, Vol. 38, pp. 775-814.

[Ste94] Stengel, R. F., 1994. Optimal Control and Estimation, Courier Corporation.

[SuB18] Sutton, R., and Barto, A. G., 2018. Reinforcement Learning, 2nd Edition, 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:1912.08957.

[Swo69] Sworder, D., 1969. "Feedback Control of a Class of Linear Systems with Jump Parameters," IEEE Trans. on Automatic Control, Vol. 14, pp. 914.

[SzL06] Szita, I., and Lorinz, A., 2006. "Learning Tetris Using the Noisy Cross-Entropy Method," Neural Computation, Vol. 18, pp. 2936-2941.

[Sze10] Szepesvari, C., 2010. Algorithms for Reinforcement Learning, Morgan and Claypool Publishers, San Franscisco, CA.

[Sze11] Szepesvari, C., 2011. "Least Squares Temporal Difference Learning and Galerkin?s Method," Presentation at the Mini-Workshop: Mathematics of Machine Learning, Mathematisches Forschungsinstitut Oberwolfach.

[TBP21] Tuncel, Y., Bhat, G., Park, J., and Ogras, U., 2021. "ECO: Enabling Energy-Neutral IoT Devices through Runtime Allocation of Harvested Energy," 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:1602.05310.

[TaL20] Tanzanakis, A., and Lygeros, J., 2020. "Data-Driven Control of Unknown Systems: A Linear Programming Approach," 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.

[Tol89] Tolwinski, B., 1989. "Newton-Type Methods for Stochastic Games," in Basar T. S., and Bernhard P. (eds), Differential Games and Applications, Lecture Notes in Control and Information Sciences, vol. 119, Springer, pp. 128-144.

[TsV96] Tsitsiklis, J. N., and Van Roy, B., 1996. "Feature-Based Methods for Large-Scale Dynamic Programming," Machine Learning, Vol. 22, pp. 59-94.

[Tse98] Tseng, P., 1998. "Incremental Gradient(-Projection) Method with Momentum Term and Adaptive Stepsize Rule," SIAM J. on Optimization, Vol. 8, pp. 506-531.

[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.

[VLK21] Vaswani, S., Laradji, I. H., Kunstner, F., Meng, S. Y., Schmidt, M., and Lacoste-Julien, S., 2021. "Adaptive Gradient Methods Converge Faster with Over-Parameterization (but you should do a line search)," arXiv:2006.06835.

[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.

[VaH20] Van Engelen, J. E., and Hoos, H. H., 2020. "A survey on Semi-Supervised Learning," Machine Learning, Vol. 109, pp. 373-440.

[Van76] Van Nunen, J. A., 1976. Contracting Markov Decision Processes, Mathematical Centre Report, Amsterdam.

[Van78] van der Wal, J., 1978. "Discounted Markov Games: Generalized Policy Iteration Method," J. of Optimization Theory and Applications, Vol. 25, pp. 125-138.

[VeM23] Vertovec, N., and Margellos, K., 2023. "State Aggregation for Distributed Value Iteration in Dynamic Programming," arXiv preprint arXiv:2303.10675.

[Vit67] Viterbi, A. J., 1967. "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Trans. on Info. Theory, Vol. IT-13, pp. 260-269.

[WCG02] Wu, G., Chong, E. K. P., and Givan, R. L., 2002. "Burst-Level Congestion Control Using Hindsight Optimization," IEEE Transactions 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.

[WGB22] Wang, T. T., Gleave, A., Belrose, N., Tseng, T., Miller, J., Dennis, M. D., Duan, Y., Pogrebniak, V., Levine, S., and Russell, S., 2022. "Adversarial Policies Beat Professional-Level Go AIs," arXiv preprint arXiv:2211.00241.

[WGP23] Weber, J., Giriyan, D., Parkar, D., Richa, A., Bertsekas, D., "Distributed Online Rollout for Multivehicle Routing in Unmapped Environments," arXiv preprint arXiv:2305.11596v1.

[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.

[WaB14] Wang, M., and Bertsekas, D. P., 2014. "Incremental Constraint Projection Methods for Variational Inequalities," Mathematical Programming, pp. 1-43.

[WaB16] Wang, M., and Bertsekas, D. P., 2016. "Stochastic First-Order Methods with Random Constraint Projection," SIAM Journal on Optimization, Vol. 26, pp. 681-717.

[WaP17] Wang, J., and Paschalidis, I. C., 2017. "An Actor-Critic Algorithm with Second-Order Actor and Critic," IEEE Trans. on Automatic Control, Vol. 62, pp. 2689-2703.

[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.

[WeV17] Westhead, D. R., and Vijayabaskar, M. S., 2017. Hidden Markov Models, Springer, Berlin.

[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.

[Whi82] Whittle, P., 1982. Optimization Over Time, Wiley, N. Y., Vol. 1, 1982, Vol. 2, 1983.

[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.

[Wil92] Williams, R. J., 1992. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," Machine Learning, Vol. 8, pp. 229-256.

[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 Journal 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.

[Won70] Wonham, W. M., 1970. "Random Differential Equations in Control Theory," Probabilistic Analysis and Related Topics, Vol. 2, pp. 131212.

[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.

[YXK24] Yilmaz, M. B., Xiang, L. and Klein, A., 2024. "Joint Beamforming and Trajectory Optimization for UAV-Aided ISAC with Dipole Antenna Array," Report.

[YYM20] Yu, L., Yang, H., Miao, L., and Zhang, C., 2019. "Rollout Algorithms for Resource Allocation in Humanitarian Logistics," IISE Transactions, Vol. 51, pp. 887-909.

[Yar17] Yarotsky, D., 2017. "Error Bounds for Approximations with Deep ReLU Networks," Neural Networks, Vol. 94, pp. 103-114.

[YuB04] Yu, H., and Bertsekas, D. P., 2004. "Discretized Approximations for POMDP with Average Cost," Proc. of the 20th Conference on Uncertainty in Artificial Intelligence, Banff, Canada.

[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.

[YuB10] Yu, H., and Bertsekas, D. P., 2010. "Error Bounds for Approximations from Projected Linear Equations," Math. of Operations Research, Vol. 35, pp. 306-329.

[YuB13] 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.

[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:1911.01004.

[Yu05] Yu, H., 2005. "A Function Approximation Approach to Estimation of Policy Gradient for POMDP with Structured Policies," Proc. of the 21st Conference on Uncertainty in Artificial Intelligence, Edinburgh, Scotland.

[Yu14] Yu, H., 2014. "Stochastic Shortest Path Games and Q-Learning," arXiv:1412.8570.

[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: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.

[ZLZ24] Zhang, Q., Liu, Y., Zhang, B., and Huang, H. Z., 2024. "Selective Maintenance Optimization Under Limited Maintenance Capacities: A Machine Learning-Enhanced Approximate Dynamic Programming," IEEE Transactions on Reliability.

[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.

[ZYB21] Zhang, K., Yang, Z. and Basar, T., 2021. "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," Handbook of Reinforcement Learning and Control, pp. 321-384.

[ZhG22] Zhu, X., and Goldberg, A. B., 2022. Introduction to Semi-Supervised Learning, Springer.

[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.