Class Notes for ASU Course CSE 691; Spring 2022 Topics in Reinforcement Learning

Based on the Books

Reinforcement Learning and Optimal Control, 2019,

Rollout, Policy Iteration, and
Distributed Reinforcement Learning, 2020,

and

Lessons from AlphaZero for Optimal,
Model Predictive, and Adaptive Control, 2022

by

Dimitri P. Bertsekas
Arizona State University

dimitrib@mit.edu

Version of February 18, 2022

Preface

These class notes (and the related slides that support them) are made available as instructional material for the 2022 CSE 691 ASU class on Reinforcement Learning. They are posted on the internet, and can be used freely for instructional purposes, provided they are not excerpted, shortened, or altered in any way.

The purpose of the notes is to provide an entry point to reinforcement learning, largely from a decision, control, and optimization point of view. They have limited scope, but they provide enough background for starting to read literature in the field and for making a choice for a research-oriented term paper. They roughly cover the material of the first six to seven lectures. They also provide the foundation for much of the material of the remaining lectures, which address topics such as parametric cost function and policy approximations (possibly involving neural networks), approximation in policy space, and aggregation, and deal much more broadly with infinite horizon problems.

To repeat, the notes (and associated slides) are not a textbook, they just provide an entry point to the field, often uneven in coverage and insufficient for in-depth understanding. At the same time the notes are at the forefront of current research, and in part discuss new and as yet unpublished material, which is also covered in my recent textbooks.

Your comments, corrections, and suggestions for improved presentation are most welcome. The contents of the notes will be periodically updated and corrected, so try to read the latest version and please refer to the version date.

The aim of the 2022 version of the ASU course is to provide an overview of the Reinforcement Learning (RL) methodology, and to focus attention on a new conceptual framework, which aims to bridge the gaps between the artificial intelligence, control theory, and operations research views of the subject. This framework centers on approximate forms of Dynamic Programming (DP), which are motivated from 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 for solution of 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 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.

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. This synergy can be understood in terms of abstract models of infinite horizon DP and simple geometrical constructions, and helps to explain the all-important stability issues within the MPC context.

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.

In these notes we will aim to explain (often with visualization) the beneficial effects of on-line decision making on top of off-line training. In the process, we will bring out the strong connections between the artificial intelligence view of RL, the control theory views of MPC and adaptive control, and the operations research view of discrete optimization algorithms. Moreover, we will describe a broad variety of algorithms (especially truncated rollout, but also other methods) that can be used for on-line play.

We will also aim to show, through the algorithmic ideas of Newton's method and the unifying principles of abstract DP, that the AlphaZero/TD-Gammon 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. Moreover, we will show that in addition to MPC and adaptive control, our conceptual framework can be effectively integrated with other important methodologies such as multiagent systems and decentralized control, discrete and Bayesian optimization, and heuristic algorithms for discrete optimization.

We finally note that while we will deemphasize mathematical proofs in these notes, there is considerable related analysis, which supports our conclusions and can be found in the author's recent RL and DP books. These books also contain additional material on closely related methodology, such as the off-line training of neural networks, on the use of policy gradient methods for approximation in policy space, and on aggregation, which will be prominent in the second half of the course, but is not covered in sufficient detail in the present class notes.

Sources

While these notes are focused primarily on on-line play, the algorithmic aspects of off-line training will be covered at length in class, and are also discussed in the author's 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.

In particular, our class discussion will include the training of neural networks for approximation in value space, the methodology of policy gradient methods for approximation in policy space, and an overview of a variety of approximation architectures, including aggregation.

The books [1] and [2] above also include a far more detailed discussion of MPC, adaptive control, and discrete optimization topics, than the present class notes. Moreover, the monograph

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

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 class notes. It also provides the mathematical foundation for the more visually oriented book

[4] 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)

which focuses in greater detail than the present class notes on the off-line training/on-line play/Newton's method conceptual framework, as well as on model predictive and adaptive control, and associated issues of stability.

All of these books are also available as ebooks; see the Athena Scientific website.

These class notes can be fruitfully supplemented by the extensive textbook and research monograph literature. This literature, is summarized in Section 1.8, and includes several accounts of reinforcement learning, based on alternative viewpoints of artificial intelligence, control theory, and operations research.

CONTENTS

1.1. AlphaZero, Off-Line Training, and On-Line Play . . . . . p. 3
1.2. Deterministic Dynamic Programming . . . . . p. 8
1.2.1. Finite Horizon Problem Formulation . . . . . p. 8
1.2.2. The Dynamic Programming Algorithm . . . . . p. 12
1.2.3. Approximation in Value Space and Rollout . . . . . p. 23
1.3. Stochastic Dynamic Programming . . . . . p. 27
1.3.1. Finite Horizon Problems . . . . . p. 27
1.3.2. Approximation in Value Space for Stochastic DP . . . p. 34
1.4. Infinite Horizon Problems - An Overview . . . . . p. 37
1.4.1. Infinite Horizon Methodology . . . . . p. 39
1.4.2. Approximation in Value Space . . . . . p. 42
1.5. Infinite Horizon Linear Quadratic Problems . . . . . p. 48
1.5.1. Visualizing Approximation in Value Space - . . . . . Newton's Method . . . . . p. 54
1.5.2. Rollout and Policy Iteration . . . . . p. 60
1.6. Examples, Variations, and Simplifications . . . . . p. 63
1.6.1. A Few Words About Modeling . . . . . p. 63
1.6.2. Problems with a Termination State . . . . . p. 65
1.6.3. State Augmentation, Time Delays, Forecasts, and . . . . . Uncontrollable State Components . . . . . p. 68
1.6.4. Partial State Information and Belief States . . . . . p. 74
1.6.5. Multiagent Problems and Multiagent Rollout . . . . . p. 77
1.6.6. Problems with Unknown Parameters - Adaptive . . . . . and Model Predictive Control . . . . . p. 82
1.7. Reinforcement Learning and Optimal Control - Some . . . . . Terminology . . . . . p. 94
1.8. Notes, Sources, and Exercises . . . . . p. 96

2. Principles of Approximation in Value Space . . . . . .

2.1. Approximation in Value and Policy Space . . . . . p. 114
2.1.1. Approximation in Value Space - One-Step and . . . . . Multistep Lookahead . . . . . p. 115
2.1.2. Approximation in Policy Space . . . . . p. 119
2.1.3. Combined Approximation in Value and . . . . . Policy Space . . . . . p. 120
2.2. Off-Line Training, On-Line Play, and Newton's Method . . p. 125
2.2.1. Approximation in Value Space and Newton's . . . . . Method . . . . . p. 132
2.2.2. Region of Stability . . . . . p. 136

2.2.3. Policy Iteration, Rollout, and Newton's Method . . . p. 141
2.2.4. How Sensitive is On-Line Play to the Off-Line Training Process? . . . p. 147
2.2.5. Why Not Just Train a Policy Network and Use it Without On-Line Play? . . . p. 149

References . . . p. 151

1

Exact and Approximate Dynamic Programming

Contents

1.1. AlphaZero, Off-Line Training, and On-Line Play . . . . p. 3
1.2. Deterministic Dynamic Programming . . . . . p. 8
1.2.1. Finite Horizon Problem Formulation . . . . . p. 8
1.2.2. The Dynamic Programming Algorithm . . . . . p. 12
1.2.3. Approximation in Value Space and Rollout . . . . p. 23
1.3. Stochastic Dynamic Programming . . . . . p. 27
1.3.1. Finite Horizon Problems . . . . . p. 27
1.3.2. Approximation in Value Space for Stochastic DP . p. 34
1.4. Infinite Horizon Problems - An Overview . . . . . p. 37
1.4.1. Infinite Horizon Methodology . . . . . p. 39
1.4.2. Approximation in Value Space . . . . . p. 42
1.5. Infinite Horizon Linear Quadratic Problems . . . . . p. 48
1.5.1. Visualizing Approximation in Value Space - Newton's Method . . . . . p. 54
1.5.2. Rollout and Policy Iteration . . . . . p. 60
1.6. Examples, Variations, and Simplifications . . . . . p. 63
1.6.1. A Few Words About Modeling . . . . . p. 63
1.6.2. Problems with a Termination State . . . . . p. 65
1.6.3. State Augmentation, Time Delays, Forecasts, and Uncontrollable State Components . . . . . p. 68

1.6.4. Partial State Information and Belief States . . . . p. 74
1.6.5. Multiagent Problems and Multiagent Rollout . . . p. 77
1.6.6. Problems with Unknown Parameters - Adaptive . . .
and Model Predictive Control . . . . . p. 82
1.7. Reinforcement Learning and Optimal Control - Some . . . .
Terminology . . . . . p. 94
1.8. Notes, Sources, and Exercises . . . . . p. 96

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

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

We will also discuss selectively in this chapter some major algorithmic topics in approximate DP and reinforcement learning (RL), including rollout and policy iteration, multiagent problems, and distributed algorithms. A broader discussion of DP/RL may be found in the author's RL books [Ber19a], [Ber20a], the DP textbooks [Ber12], [Ber17], [Ber18a], the neuro-dynamic programming monograph [BeT96], as well as the textbook literature described in the last section of this chapter.

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

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

One of the most exciting recent success stories in RL is the development of the AlphaGo and AlphaZero programs by DeepMind Inc; see [SHM16], [SHS17], [SSS17]. AlphaZero plays Chess, Go, and other games, and is an improvement in terms of performance and generality over AlphaGo, which plays the game of Go only. Both programs play better than all competitor computer programs available in 2021, 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 notes, and is portable to far broader realms of engineering, economics, and other fields. This is the methodology of DP, policy iteration, limited lookahead, rollout, and approximation in value space.†

To understand the overall structure of AlphaZero, and 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 possible moves at any given chess position (these are the probabilities with which the player selects the possible moves at the given position). A position evaluator assigns a numerical score to any given chess position (akin to a "probability" of winning the game from that position), and thus predicts quantitatively the performance of a player starting from any position. The chess player and the position evaluator are represented by two neural networks, a *policy network* and a *value network*, which accept a chess position and generate a set of move probabilities and a position evaluation, respectively.‡

† 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]. 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.

‡ Here the neural networks play the role of *function approximators*. By viewing a player as a function that assigns move probabilities to a position, and a position evaluator as a function that assigns a numerical score to a position, the policy and value networks provide approximations to these functions based on

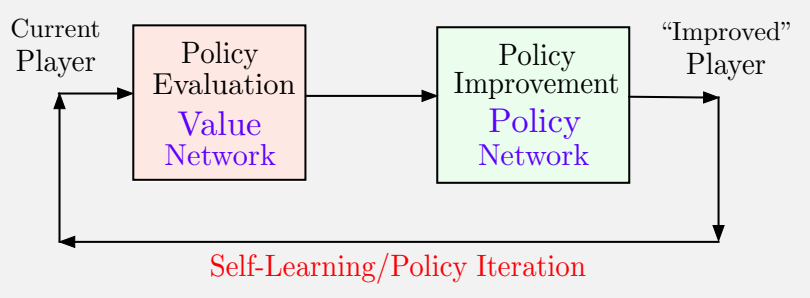

Figure 1.1.1 Illustration of the AlphaZero training algorithm. It generates a sequence of position evaluators and chess players. The position evaluator and the chess player are represented by two neural networks, a value network and a policy network, which accept a chess position and generate a position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

In the more conventional DP-oriented terms of these notes, 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 DP algorithm that will be of primary interest to us in these notes. Starting from a given player, it repeatedly generates (approximately) improved players, and settles on a final player that is judged empirically to be "best" out of all the players generated.‡ Policy iteration may be separated conceptually in two stages (see Fig. 1.1.1).

(a) *Policy evaluation*: Given the current player and a chess position, the

training with data (training algorithms for neural networks and other approximation architectures are discussed in the books [Ber19a], [Ber20a]).

† 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 program to play against a known fixed opponent; this is a one-player setting.

‡ 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."

outcome of a game played out from the position provides a single data point. Many data points are thus collected, and are used to train a value network, whose output serves as the position evaluator for that player.

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

On-Line Play and Approximation in Value Space - Rollout

Suppose that a "final" player has been obtained through the AlphaZero off-line training process just described. It could then be used in principle to play chess against any human or computer opponent, since it is capable of generating move probabilities at each given chess position using its policy network. In particular, during on-line play, at a given position the player can simply choose the move of highest probability supplied by the off-line trained policy network. This player would play very fast on-line, but it would not play good enough chess to beat strong human opponents. The extraordinary strength of AlphaZero is attained only after the player and its position evaluator obtained from off-line training have been embedded into another algorithm, which we refer to as the "on-line player." Given the policy network/player obtained off-line and its value network/position evaluator, this algorithm plays as follows (see Fig. 1.1.2).

At a given position, it generates a lookahead tree of all possible multiple move and countermove sequences, up to a given depth. It then runs the off-line obtained player for some more moves, and then evaluates the effect of the remaining moves by using the position evaluator of the off-line obtained value network. Actually the middle portion, called "truncated rollout," is not used in the published version of AlphaZero/chess [SHS17], [SHS17]; the first portion (multistep lookahead) is quite long and implemented efficiently, so that the rollout portion is not essential. Rollout is used in AlphaGo [SHM16], and plays a very important role in Tesauro's backgammon program [TeG96]. The reason is that in backgammon, long

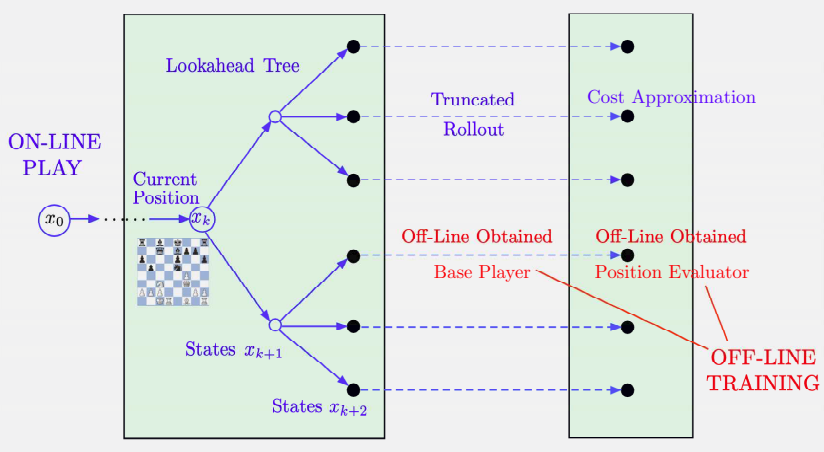

Figure 1.1.2 Illustration of an on-line player such as the one used in AlphaGo, AlphaZero, and Tesauro's backgammon program [TeG96]. At a given position, it generates a lookahead tree of multiple moves up to some depth, then runs the off-line obtained player for some more moves, and evaluates the effect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

multistep lookahead is not possible because of rapid expansion of the lookahead tree with every move.†

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

Note that these two processes may be decoupled and may be designed independently. For example the off-line training portion may be very simple, such as using a simple 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.

† Tesauro's rollout-based backgammon program [TeG96] does not use a policy network. It involves only a value network trained by his earlier TD-Gammon algorithm [Tes94] (an approximate policy iteration algorithm), which is used to generate moves for the truncated rollout via a one-step or two-step lookahead minimization. The position evaluation used at the end of the truncated rollout is also provided by the value network.

In control system design, similar architectures to the ones of AlphaZero and TD-Gammon are employed in model predictive control (MPC). There, the number of steps in lookahead 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 within this context. We will discuss later the structure of MPC and its similarities with the AlphaZero architecture.

Dynamic programming frameworks with cost function approximations that are similar to the on-line player illustrated in Fig. 1.1.2, are also known as *approximate dynamic programming*, or *neuro-dynamic programming*, and will be central for our purposes. They will be generically referred to as *approximation in value space* in these notes.‡

1.2 DETERMINISTIC DYNAMIC PROGRAMMING

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

1.2.1 Finite Horizon Problem Formulation

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

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

where

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

‡ 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 not be discussed at any length here, but an account that is consistent in terminology with these notes may be found in the RL book [Ber19a].

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.

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

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

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

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

† 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 Chapter 5). Unfortunately, this notation is not well suited for deterministic models, and also for continuous spaces models, both of which are important for the purposes of this book. For the latter models, it involves transition probability distributions over continuous spaces, and leads to mathematics that are far more complex as well as less intuitive than those based on the use of the system function (1.1).

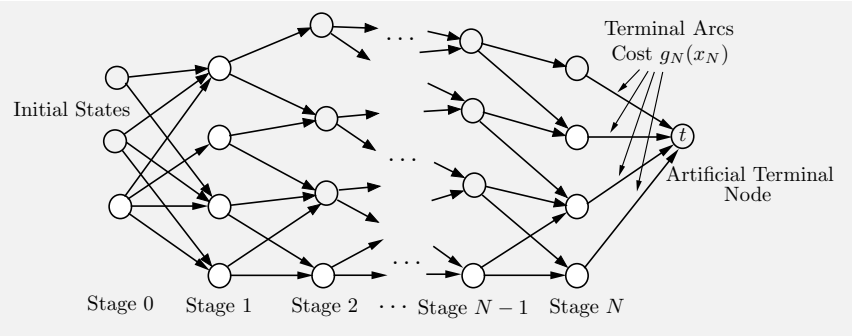

Figure 1.2.2 Transition graph for a deterministic finite-state system. Nodes correspond to states x_k. Arcs correspond to state-control pairs (x_k, u_k). An arc (x_k, u_k) has start and end nodes x_k and $x_{k+1} = f_k(x_k, u_k)$, respectively. We view the cost $g_k(x_k, u_k)$ of the transition as the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

obtaining the optimal value as a function of x_0:†

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

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*

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

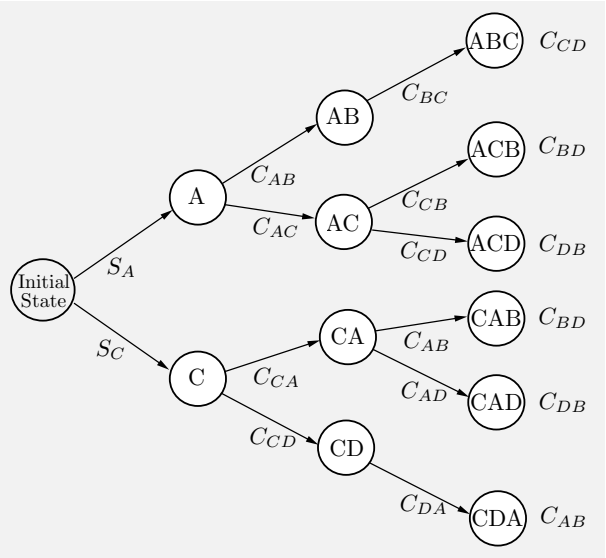

Figure 1.2.3 The transition graph of the deterministic scheduling problem of Example 1.2.1. Each arc of the graph corresponds to a decision leading from some state (the start node of the arc) to some other state (the end node of the arc). The corresponding cost is shown next to the arc. The cost of the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

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 is to break down the solution into components, which can be computed sequentially. The following is an illustrative example.

Example 1.2.1 (A Deterministic Scheduling Problem)

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

† 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 [Ber17], Section 2.1, and [Ber91], [Ber98] for extensive accounts of shortest path methods, which connect with our discussion here.

sequence is the sum of the setup costs associated with it; for example, the operation sequence ACDB has cost $S_A + C_{AC} + C_{CD} + C_{DB}$.

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

1.2.2 The Dynamic Programming Algorithm

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

Principle of Optimality

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

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

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

The subproblem referred to above is called the *tail subproblem* that starts at x_k^*. Stated succinctly, the principle of optimality says that *the tail of an optimal sequence is optimal for the tail subproblem*. Its intuitive justification is simple. If the truncated control sequence $\{u_k^*, \ldots, u_{N-1}^*\}$ were not optimal as stated, we would be able to reduce the cost further by switching to an optimal sequence for the subproblem once we reach x_k^* (since the preceding choices of controls, $u_0^*, \ldots, u_{k-1}^*$, do not restrict our future choices).

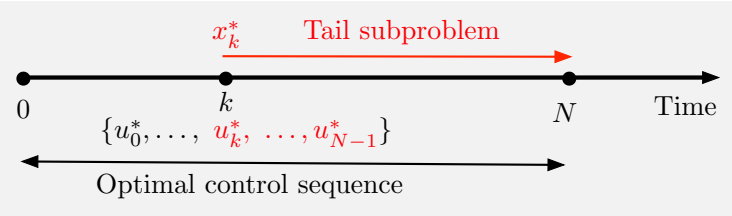

Figure 1.2.4 Schematic illustration of the principle of optimality. The tail $\{u_k^*, \ldots, u_{N-1}^*\}$ of an optimal sequence $\{u_0^*, \ldots, u_{N-1}^*\}$ is optimal for the tail subproblem that starts at the state x_k^* of the optimal state trajectory.

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

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

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

Example 1.2.1 (Scheduling Problem - Continued)

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

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

† 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."

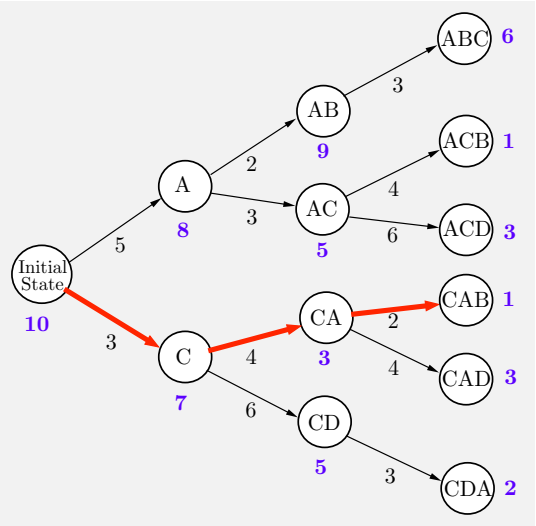

Figure 1.2.5 Transition graph of the deterministic scheduling problem, with the cost of each decision shown next to the corresponding arc. Next to each node/state we show the cost to optimally complete the schedule starting from that state. This is the optimal cost of the corresponding tail subproblem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

solve all possible tail subproblems of length two, then all tail subproblems of length three, and finally the original problem that has length four (the subproblems of length one are of course trivial because there is only one operation that is as yet unscheduled). As we will see shortly, the tail subproblems of length $k+1$ are easily solved once we have solved the tail subproblems of length k, and this is the essence of the DP technique.

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

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

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

State CA: Here the possibilities are to (a) schedule operation B and then D, which has cost 3, or (b) schedule operation D and then B, which has cost 7. The first possibility is optimal, and the corresponding cost of

the tail subproblem is 3, as shown next to node CA in Fig. 1.2.5.

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

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

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

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

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

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

Finding an Optimal Control Sequence by DP

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

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

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

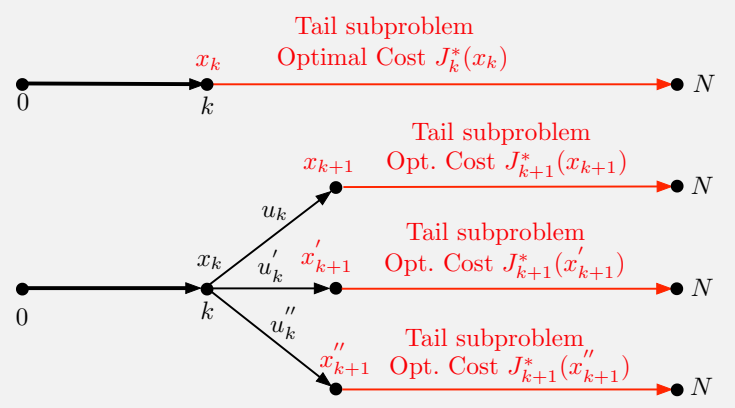

Figure 1.2.6 Illustration of the DP algorithm. The tail subproblem that starts at x_k at time k minimizes over $\{u_k, \ldots, u_{N-1}\}$ the "cost-to-go" from k to N,

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

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

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

DP Algorithm for Deterministic Finite Horizon Problems

Start with

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

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 optimal cost-to-go functions $J_k^*(x_k)$ are illustrated in Fig. 1.2.6. Note that at stage k, the calculation in Eq. (1.5) 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.

DP Algorithm for General Discrete Optimization Problems

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

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 (Example 1.2.3) 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

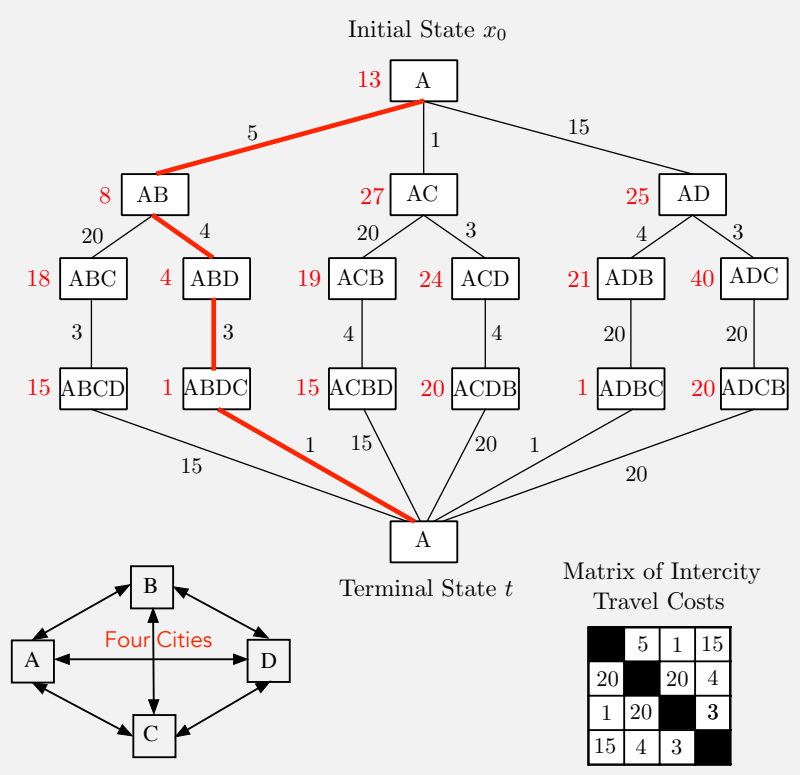

Figure 1.2.7 Example of a DP formulation of the traveling salesman problem. The travel times between the four cities A, B, C, and D are shown in the matrix at the bottom. We form a graph whose nodes are the k-city sequences and correspond to the states of the kth stage, assuming that A is the starting city. The transition costs/travel times are shown next to the arcs. The optimal costs-to-go are generated by DP starting from the terminal state and going backwards towards the initial state, and are shown next to the nodes. There is a unique optimal sequence here (ABDCA), and it is marked with thick lines. The optimal sequence can be obtained by forward minimization [cf. Eq. (1.8)], starting from the initial state x_0.

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 traveling salesman and related scheduling problems are typically addressed with approximation methods, some of which are based on DP, and will be discussed in future chapters.

Let us now extend the ideas of the preceding example to the general discrete optimization problem:

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

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

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

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

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

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

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

Let

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

denote the optimal cost starting from the k-solution $(u_0, \ldots, u_{k-1})$, i.e., the optimal cost of the problem over solutions whose first k components

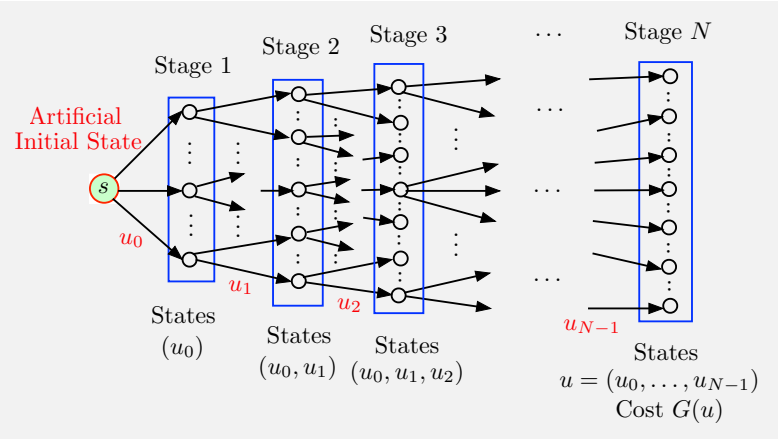

Figure 1.2.8. Formulation of a discrete optimization problem as a DP problem with N stages. There is a cost $G(u)$ only at the terminal stage on the arc connecting an N-solution $u = (u_0, \ldots, u_{N-1})$ upon reaching the terminal state. Alternative formulations may use fewer states by taking advantage of the problem's structure.

are constrained to be equal to $u_0, \ldots, u_{k-1}$. The DP algorithm is described by the equation

$$J_k^*(u_0, \ldots, u_{k-1}) = \min_{u_k \in U_k(u_0, \ldots, u_{k-1})} J_{k+1}^*(u_0, \ldots, u_{k-1}, u_k), \tag{1.9}$$

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

$$u_k^* \in \arg \min_{u_k \in U_k(u_0^*, \ldots, u_{k-1}^*)} J_{k+1}^*(u_0^*, \ldots, u_{k-1}^*, u_k), \quad k = 0, \ldots, N-1, \tag{1.10}$$

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 (1.10) as a starting point for the use of approximation methods. For example we may try to use approximation in value space, whereby we replace J_{k+1}^* with some suboptimal $\tilde{J}_{k+1}$ in Eq. (1.10). One possibility is to use as

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

the cost generated by a heuristic method that solves the problem suboptimally with the values of the first $k+1$ decision components fixed at $u_0^*, \ldots, u_{k-1}^*, u_k$. This is the *rollout algorithm*, which is a very simple and effective approach for approximate combinatorial optimization; see the next section.

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

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

$$Q_k^*(x_k, u_k) = g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big). \tag{1.11}$$

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

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} Q_k^*(x_k, u_k). \tag{1.12}$$

Moreover, the DP algorithm (1.5) can be written in an essentially equivalent form that involves Q-factors only [cf. Eqs. (1.11)-(1.12)]:

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

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*. We will discuss various forms of Q-learning later.

† 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.

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 procedure (1.8).

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

$$\tilde{u}_k \in \arg\min_{u_k \in U_k(\tilde{x}_k)} \Big[g_k(\tilde{x}_k, u_k) + \tilde{J}_{k+1}\big(f_k(\tilde{x}_k, u_k)\big) \Big], \tag{1.13}$$

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 (1.13) 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

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

the system, and not for every state within the potentially enormous state space, as is the case for exact DP.

The algorithm (1.13) is said to involve a *one-step lookahead minimization*, since it solves a one-stage DP problem for each k. In the next chapter we will also discuss the possibility of *multistep lookahead*, which involves the solution of an ℓ-step DP problem, where ℓ is an integer, $1 < \ell < N-k$, with a terminal cost function approximation $\tilde{J}_{k+\ell}$. Multistep lookahead typically (but not always) provides better performance over one-step lookahead in RL approximation schemes, and will be discussed in Chapter 2. For example in Alphazero chess, long multistep lookahead is critical for good on-line performance. The intuitive reason is that with ℓ stages being treated "exactly" (by optimization), the effect of the approximation error

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

tends to become less significant as ℓ increases. However, the solution of the multistep lookahead optimization problem, instead of the one-step lookahead counterpart of Eq. (1.13), becomes more time consuming.

Rollout, Cost Improvement, and On-Line Replanning

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

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

There are also several variants of rollout, including versions involving multiple heuristics, combinations with other forms of approximation

† For an intuitive justification of the cost improvement mechanism, note that the rollout control $\tilde{u}_k$ is calculated from Eq. (1.13) 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.

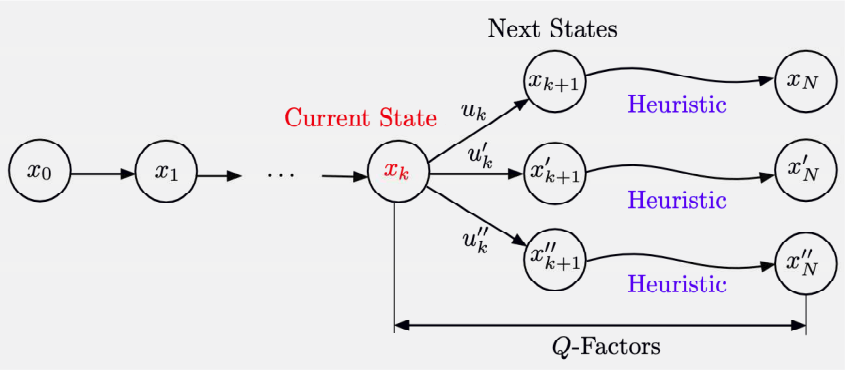

Figure 1.2.9 Schematic illustration of rollout with one-step lookahead for a deterministic problem. At state x_k, for every pair (x_k, u_k), $u_k \in U_k(x_k)$, the base heuristic generates an approximate Q-factor

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

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

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:

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

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

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k), \tag{1.14}$$

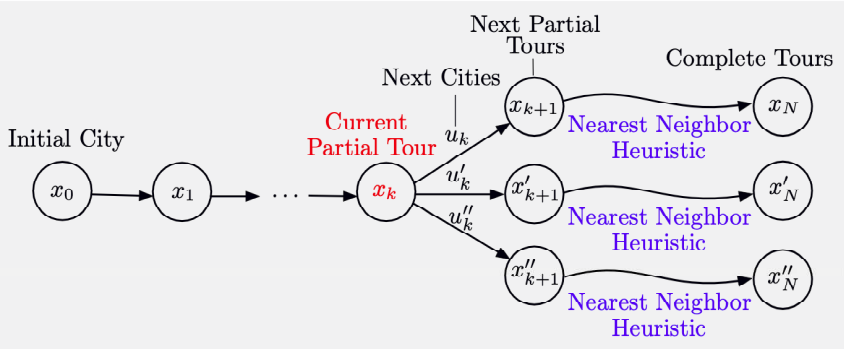

Figure 1.2.10 Rollout with the nearest neighbor heuristic for the traveling salesman problem of Example 1.2.3. The initial state x_0 consists of a single city. The final state x_N is a complete tour of N cities, containing each city exactly once.

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

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big); \tag{1.15}$$

see Fig. 1.2.9. Rollout defines a suboptimal policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, referred to as the *rollout policy*, where for each x_k and k, $\tilde{\mu}_k(x_k)$ is the control produced by the Q-factor minimization (1.14).

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 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 $\{c_0, c_1, \ldots, c_k\}$ 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

$$g(c_0, c_1) + \cdots + g(c_{N-2}, c_{N-1}) + g(c_{N-1}, c_0). \tag{1.16}$$

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. (1.16). 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 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 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 counterpat primarily in the nature of the discrete-time dynamic system that governs the evolution of the state x_k. This system includes a random

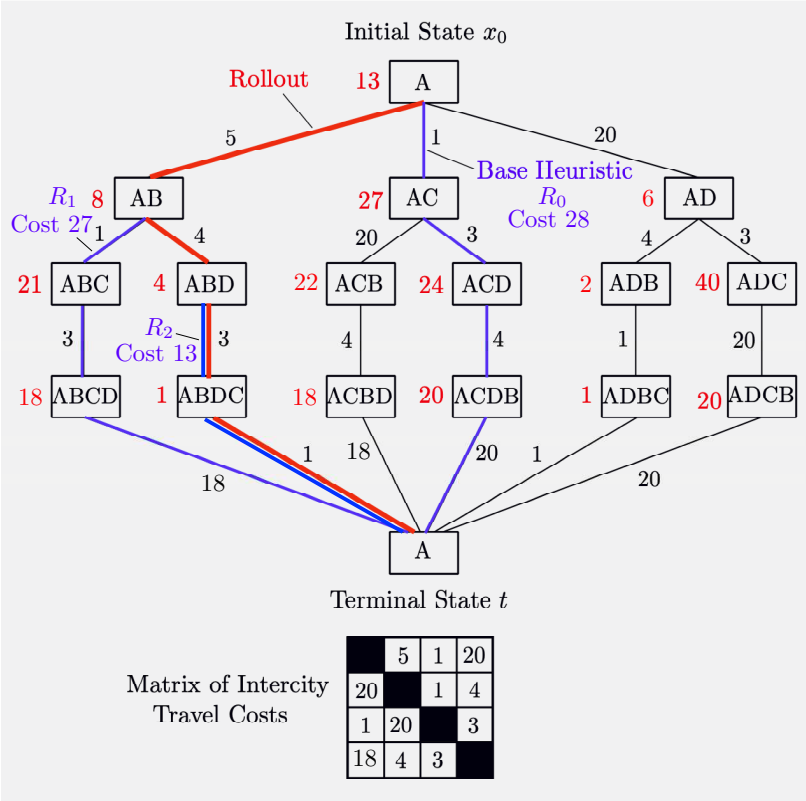

Figure 1.2.11 A traveling salesman problem example of rollout with the nearest neighbor base heuristic. At city A, the nearest neighbor heuristic generates the tour ACDBA (labelled R_0). At city A, the rollout algorithm compares the tours ABCDA, ACDBA, and ADCBA, finds ABCDA (labelled R_1) to have the least cost, and moves to city B. At AB, the rollout algorithm compares the tours ABCDA and ABDCA, finds ABDCA (labelled R_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). Note that the algorithm generates three tours/solutions, R_0, R_1, and R_2, of decreasing costs 28, 27, and 13, respectively. The first tour R_0 is generated by the base heuristic starting from the initial state, while the last tour R_2 is generated by rollout. This is suggestive of a general result (see the RL books [Be19a], [Ber20a]): the rollout algorithm for deterministic problems generates a sequence of solutions of decreasing cost. In particular, the final rollout solution is no worse in terms of cost than the base heuristic solution. In this example, the tour R_2 generated by rollout is optimal, but this is just a coincidence.

"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

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

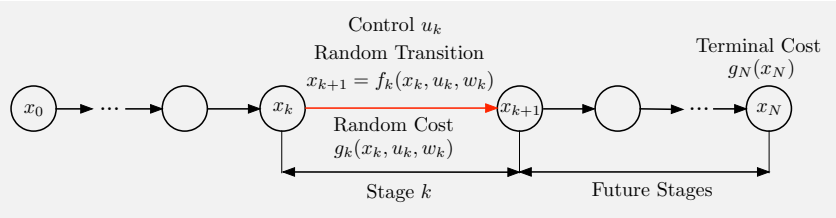

Figure 1.3.1 Illustration of an N-stage stochastic optimal control problem. Starting from state x_k, the next state under control u_k is generated randomly, according to $x_{k+1} = f_k(x_k, u_k, w_k)$, where w_k is the random disturbance, and a random stage cost $g_k(x_k, u_k, w_k)$ is incurred.

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

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

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

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

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

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

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

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

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

Finite Horizon Stochastic Dynamic Programming

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

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

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

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

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

DP Algorithm for Stochastic Finite Horizon Problems

Start with

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

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

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\Big\}. \tag{1.17}$$

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 [Ber17]).

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. (1.17). 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 (1.17) 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 [Ber17]).

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

$$v_{k+1} = v_k + bu_k + w_k, \tag{1.18}$$

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. (1.17), we obtain

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

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

$$J_{N-1}^*(x_{N-1}) = qx_{N-1}^2 + q\sigma^2 + \min_{u_{N-1}} \Big[ru_{N-1}^2 + q(ax_{N-1} + bu_{N-1})^2\Big]. \quad (1.19)$$

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. (1.19), we obtain with a straightforward calculation

$$J_{N-1}^*(x_{N-1}) = P_{N-1}x_{N-1}^2 + q\sigma^2,$$

where

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

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 have the optimal policy and optimal cost-to-go function in the form

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

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

where

$$L_k = -\frac{abP_{k+1}}{r + b^2 P_{k+1}}, \qquad k = 0, 1, \ldots, N-1, \tag{1.20}$$

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

$$P_k = \frac{a^2 r P_{k+1}}{r + b^2 P_{k+1}} + q, \qquad k = 0, 1, \ldots, N-1, \tag{1.21}$$

starting from the terminal condition $P_N = q$.

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 [Ber17], 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 [Ber17], 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.

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

We can define optimal Q-factors for a stochastic problem, similar to the case of deterministic problems [cf. Eq. (1.11)], as the expressions that are minimized in the right-hand side of the stochastic DP equation (1.17). They are given by

$$Q_k^*(x_k, u_k) = E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\Big\}. \tag{1.22}$$

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. (1.22) is replaced by an approximation $\tilde{J}_{k+1}$. Generally, a 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 (1.13).

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

At any state x_k encountered at stage k, set

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

Similar to the deterministic case, the one-step lookahead minimization (1.23) 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.

Our discussion of rollout of Section 1.2 also applies to stochastic problems: we select $\tilde{J}_k$ to be the cost function of a suitable base policy (perhaps with some approximation). Note that any policy can be used on-line as base policy, including policies obtained by a sophisticated off-line procedure, using for example neural networks and training data.† The rollout algorithm

† The principal role of neural networks within the context of these notes 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

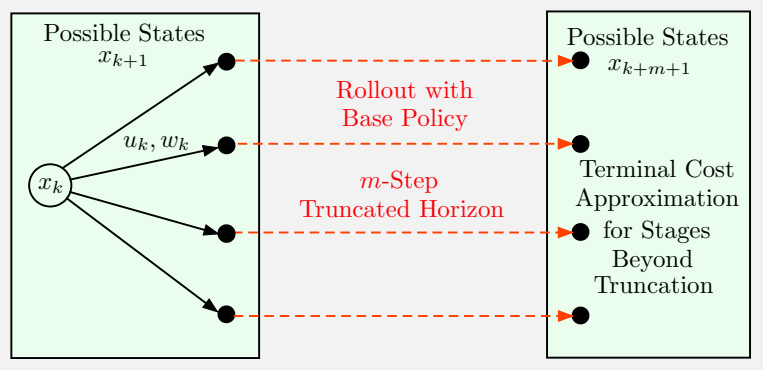

Figure 1.3.2 Schematic illustration of truncated rollout. One-step lookahead is followed by simulation of the base policy for m steps, and an approximate cost $\tilde{J}_{k+m+1}(x_{k+m+1})$ is added to the cost of the simulation, which depends on the state x_{k+m+1} obtained at the end of the rollout. If the base policy simulation is omitted (i.e., $m = 0$), one recovers the general approximation in value space scheme (1.23). There are also multistep lookahead versions of truncated rollout (see Chapter 2).

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

A major variant of rollout is *truncated rollout*, which combines the use of one-step optimization, simulation of the base policy for a certain number of steps m, and then adds an approximate cost $\tilde{J}_{k+m+1}(x_{k+m+1})$ to the cost of the simulation, which depends on the state x_{k+m+1} obtained at the end of the rollout (see Chapter 2). Note that if one foregoes the use of a base policy (i.e., $m = 0$), one recovers as a special case the general approximation in value space scheme (1.23); see Fig. 1.3.2. 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.

We may also simplify the lookahead minimization over $u_k \in U_k(x_k)$

---

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 are discussed in many sources, including the RL books [Ber19a] and [Ber20a].

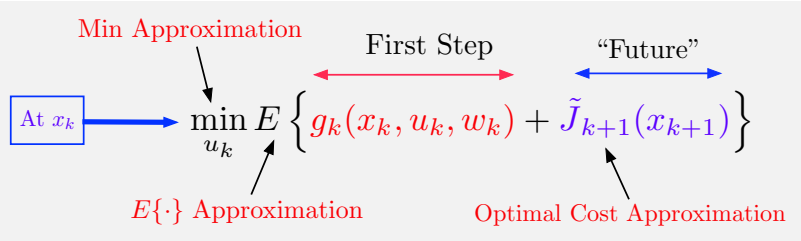

Figure 1.3.3 Schematic illustration of approximation in value space for stochastic problems, and the three approximations involved in its design. Typically the approximations can be designed independently of each other. There are also multistep lookahead versions of approximation in value space, which will be discussed later.

[cf. Eq. (1.17)]. In particular, in the multiagent case where the control consists of multiple components, $u_k = (u_k^1, \ldots, u_k^m)$, a sequence of m single component minimizations can be used instead, with potentially enormous computational savings resulting.

There is one additional issue in approximation in value space for stochastic problems: the computation of the expected value in Eq. (1.23) may be very time-consuming. Then one may consider approximations in the computation of this expected value, based for example on Monte Carlo simulation or other schemes. Some of the possibilities along this line will be discussed in the next chapter.

Figure 1.3.3 illustrates the three approximations involved in approximation in value space for stochastic problems: *cost-to-go approximation, expected value approximation, and simplified minimization*. They may be designed largely independently of each other, and with a variety of methods. Much of the discussion in these notes will revolve around different ways to organize these three approximations. Another major issue is the mechanism by which the choice of the approximations $\tilde{J}_k$ affects the cost function $J_{\tilde{\pi}}$ of the one-step lookahead policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$. We will see later that $\tilde{J}_k$ *and* $J_{\tilde{\pi}}$ *are connected through a form of Newton's method*.

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

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k). \tag{1.24}$$

We have seen that it is possible to implement approximation in value space by using cost function approximations [cf. Eq. (1.23)] or by using Q-factor approximations [cf. Eq. (1.24)], so the question arises which one to use in a given practical situation. One important consideration is the facility of obtaining suitable cost or Q-factor approximations. This depends largely on the problem and also on the availability of data on which the approximations can be based. However, there are some other major considerations.

In particular, the cost function approximation scheme

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

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

By comparison, the Q-factor function approximation scheme (1.24) 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 does 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. (1.24).

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

$$J_\pi(x_0) = \lim_{N \to \infty} \underset{\substack{w_k \\ k=0,1,\ldots}}{E} \left\{ \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k), w_k\big) \right\}; \tag{1.25}$$

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

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

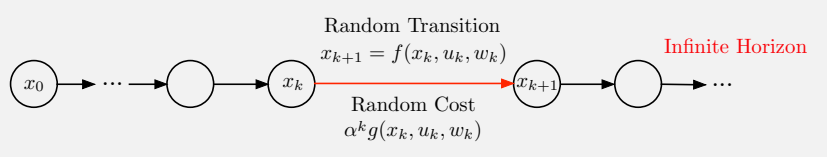

Figure 1.4.1 Illustration of an infinite horizon problem. The system and cost per stage are stationary, except for the use of a discount factor α. If $\alpha = 1$, there is typically a special cost-free termination state that we aim to reach.

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 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.

(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

$$x_{k+1} = f(x_k, u_k), \qquad k = 0, 1, \ldots, \tag{1.26}$$

and

$$J_\pi(x_0) = \lim_{N\to\infty} \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k)\big). \tag{1.27}$$

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

$$g(x, u) \geq 0, \qquad \text{for all } x \neq t,\ u \in U(x), \tag{1.28}$$

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].

An important special case is *finite-state deterministic problems*. Finite horizon versions of these problems include challenging discrete optimization problems, whose exact solution is practically impossible. It is possible to transform such problems to infinite horizon SSP problems, so that the conceptual framework developed here 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

$$J_{k+1}(x) = \min_{u\in U(x)} E_w\Big\{g(x, u, w) + \alpha J_k\big(f(x, u, w)\big)\Big\}, \qquad k = 0, 1, \ldots, \tag{1.29}$$

starting from $J_0(x) \equiv 0$.† The algorithm (1.29) is known as the *value iteration* algorithm (VI for short). Since the infinite horizon cost of a given

† 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,

policy is, by definition, the limit of the corresponding N-stage costs as $N \to \infty$, it is natural to speculate that:

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

$$J^*(x) = \lim_{N\to\infty} J_N(x) \tag{1.30}$$

for all states x.

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

$$J^*(x) = \min_{u\in U(x)} E_w\Big\{g(x,u,w) + \alpha J^*\big(f(x,u,w)\big)\Big\}. \tag{1.31}$$

This is obtained by taking the limit as $N \to \infty$ in the VI algorithm (1.29) using Eq. (1.30). The preceding equation, called *Bellman's equation*, is really a system of equations (one equation per state x), which has as solution the optimal costs-to-go of all the states.

(3) If $\mu(x)$ attains the minimum in the right-hand side of the Bellman equation (1.31) for each x, then the policy $\{\mu, \mu, \ldots\}$ should be optimal. This type of policy is called *stationary*. Intuitively, optimal policies can be found within this class of policies, since optimization of the future costs (the tail subproblem) when starting at a given state looks the same regardless of the time when we start.

All three of the preceding results hold for finite-state discounted and also 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 mathe-

---

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 (1.29).

matical proofs can be found in several sources, e.g., [Ber12], Chapter 1, or [Ber19a], Chapter 4.†

Note that the VI algorithm of Eq. (1.29) simply expresses the fact that the optimal cost for $k+1$ stages is obtained by minimizing the sum of the first stage cost and the optimal cost for the next k stages starting from the next state $f(x,u,w)$. The latter cost, however, should be discounted by α in view of the cost definition (1.25). The intuitive interpretation of the Bellman equation (1.31) is that it is the limit as $k \to \infty$ of the VI algorithm (1.29) assuming that $J_k \to J^*$.

Let us also consider stationary policies $\{\mu, \mu, \ldots\}$. For simplicity we will denote them by μ, and write J_μ for their cost functions. We expect that J_μ satisfies the Bellman equation for μ, given by

$$J_\mu(x) = E_w\Big\{g\big(x,\mu(x),w\big) + \alpha J_\mu\big(f(x,\mu(x),w)\big)\Big\}, \qquad \text{for all } x. \quad (1.32)$$

We can view this as just the Bellman equation (1.31) 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 generated by

$$J_{\mu,k+1}(x) = E_w\Big\{g\big(x,\mu(x),w\big) + \alpha J_{\mu,k}\big(f(x,\mu(x),w)\big)\Big\}, \qquad k = 0,1,\ldots, \quad (1.33)$$

starting from $J_{\mu,0}(x) \equiv 0$ or some other initial condition; cf. Eqs. (1.29)-(1.30).

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, and their variants are some of these. 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 these notes, we will only discuss approximate forms and variations of the policy iteration algorithm, which we describe next.

† For undiscounted problems and discounted problems with unbounded cost per stage, we may still adopt the three 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] and [Ber22a].

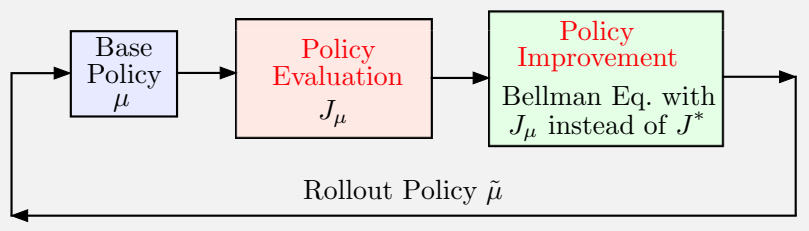

Figure 1.4.2 Schematic illustration of PI as repeated rollout. It generates a sequence of policies, with each policy μ in the sequence being the base policy that generates the next policy $\tilde{\mu}$ in the sequence as the corresponding rollout policy.

Policy Iteration

A major class of infinite horizon algorithms is based on *policy iteration* (PI for short). 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. (1.32). However, the value $J_\mu(x)$ for any x can also be computed by Monte Carlo simulation, by averaging over many randomly generated trajectories the cost of the policy starting from x. Other, more sophisticated possibilities include the use of specialized simulation-based methods, such as *temporal difference methods*, for which there is extensive literature (see e.g., the books [[BeT96], [SuB98], [Ber12]).

(b) *Policy improvement*, which computes the "improved" (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 $J_{\tilde{\mu}}(x) \leq J_\mu(x)$ for all x, as we will see later.]

We will refer to the starting and ending policies μ and $\tilde{\mu}$ as the *base* and *rollout* policies of the iteration, respectively. We will discuss several different forms of PI. We will also argue that PI is the DP concept that forms the foundation for self-learning in RL, i.e., learning from data that is self-generated (from the system itself as it operates) rather than from data supplied from an external source.

Note that *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 a rollout policy $\tilde{\mu}$. It may be viewed as one-step lookahead using J_μ as terminal cost function approximation, and 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

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*

$$\tilde{\mu}(x) \in \arg\min_{u \in U(x)} E\Big\{g(x,u,w) + \alpha \tilde{J}\big(f(x,u,w)\big)\Big\}. \tag{1.34}$$

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 (1.31) 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.

Generally, it is desirable that $J_{\tilde{\mu}}$ is close to J^* in some sense. However, for classical control problems, which involve steering the state towards a goal 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 these notes, we will focus on 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.,

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

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 later.

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 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

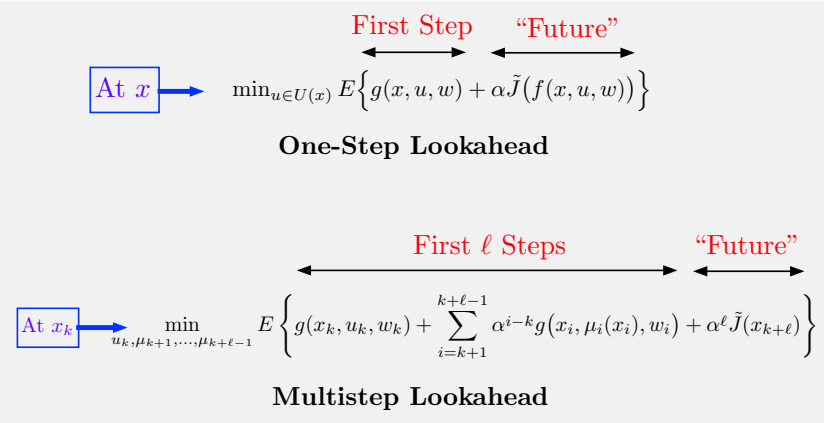

Figure 1.4.3 Schematic illustration of approximation in value space with one-step and ℓ-step lookahead minimization. In the former case, the minimization yields at state x a control $\tilde{u}$, which defines the one-step lookahead policy $\tilde{\mu}$ via

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

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 $\tilde{\mu}$.

control $\tilde{u}_k$ is applied at x_k, and defines the ℓ-step lookahead policy $\tilde{\mu}$ via $\tilde{\mu}(x_k) = \tilde{u}_k$. The sequence $\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}$ is 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, and is also supported by error bounds, given for example in the books [Ber19a], [Ber20a]. 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. On the other hand, solving the multistep lookahead minimization problem, instead of the one-step lookahead counterpart of Eq. (1.34), is more time consuming.

Let us also note that one-step and multistep lookahead minimization may be supplemented by truncated rollout, as we have discussed in Section 1.1. In one possible view of schemes like this, the effects of truncated rollout

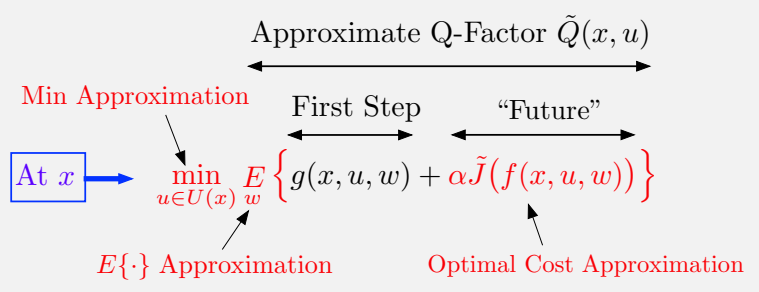

Figure 1.4.3 Schematic illustration of approximation in value space with one-step and multistep lookahead for infinite horizon problems. There are three potential areas of approximation, which can be considered independently of each other: optimal cost approximation, expected value approximation, and minimization approximation.

may be expressed through the terminal cost approximation $\tilde{J}$.

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

There are three potential areas of approximation for infinite horizon problems: optimal cost approximation, expected value approximation, and minimization approximation; cf. Fig. 1.4.3. They are similar to their finite horizon counterparts. 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 the random quantity w may be replaced by a deterministic quantity; this is known as the *certainty equivalence approximation*. A major computational advantage of deterministic problems is that they do not involve any expected values.

(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.

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

Constructing Terminal Cost Approximations

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 a sophisticated off-line training method, as discussed in Section 1.1, in connection with chess and backgammon. Alternatively, the approximate values $\tilde{J}(x)$ are 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 further details.

(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*, which is described and analyzed in the books [Ber12], [Ber19a], and the papers [Ber18b], [Ber18c]. 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 heuristic 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. This may be an important consideration for the practical implementation of rollout algorithms, particularly for stochastic problems.

(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.

Let us also mention that for problems with special structure, $\tilde{J}$ may be chosen so that the one-step lookahead minimization (1.34) is facilitated. In fact, under favorable circumstances, the lookahead minimization may be carried out in closed form. An example is when the control enters linearly in the system equation and quadratically in the cost function, while the terminal cost approximation is chosen to be quadratic.

Understanding Approximation in Value Space

We will now discuss some of our objectives as we will 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 $\tilde{\mu}$, 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 affected by the quality of the off-line training?* Here we are interested in whether $J_{\tilde{\mu}}(x)$ is smaller than $\tilde{J}(x)$ across a range of states x of interest, and by how much.

(b) *How sensitive is the quality of the lookahead policy to the quality of the off-line training?* Here we are interested to understand how much $J_{\tilde{\mu}}$ changes when $\tilde{J}$ changes across a range of interest. For example, how much should we care about improving $\tilde{J}$ through a longer and more sophisticated training process?

(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. As

noted earlier, we will use an optimization-based definition of stability, calling the lookahead policy $\tilde{\mu}$ stable if $J_{\tilde{\mu}}(x) < \infty$, for all x. An example is an SSP problem with positive cost per stage, where some policies may not guarantee that the termination state will be reached; these policies are viewed as unstable. An example of a context where there are no stability concerns is discounted problems with bounded cost per stage; here all policies are stable according to our definition. While there are several alternative definitions of stability, which may be better-matched to specific contexts, our definition of stability is suitable for the very broad class of problems that we are dealing with.

(d) *What is the region of stability?* Here we may be interested to characterize the set of terminal cost approximations $\tilde{J}$ that lead to a stable lookahead policy.

(e) *How does the length of lookahead minimization or the length of the truncated rollout affect the stability and quality of the multistep lookahead policy?* 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 the mechanism by which this occurs.

In what follows we will try to give some answers to these questions, first in the next section, in the context of simple linear quadratic problems, and then in Chapter 2 within a more general context. The monograph [Ber22b] addresses such issues in greater detail.

1.5 INFINITE HORIZON 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 nonnegative cost problem. This is the classical continuous-spaces problem where the system is linear, with no control constraints, and the cost function is quadratic. In its general form, this problem deals with the case where the system is

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

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], Chapter 3, and [Ber12], Chapter 4.

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

$$x_{k+1} = ax_k + bu_k; \tag{1.35}$$

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

$$\sum_{k=0}^{\infty}(qx_k^2 + ru_k^2), \tag{1.36}$$

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.

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

$$J^*(x) = K^*x^2, \tag{1.37}$$

where the scalar K^* solves the equation

$$K = F(K), \tag{1.38}$$

with F defined by

$$F(K) = \frac{a^2rK}{r + b^2K} + q. \tag{1.39}$$

This is the limiting form of Eq. (1.21).

Moreover, the optimal policy is linear of the form

$$\mu^*(x) = L^*x, \tag{1.40}$$

where L^* is the scalar given by

$$L^* = -\frac{abK^*}{r + b^2K^*}. \tag{1.41}$$

For justification of Eqs. (1.38)-(1.41), we show that J^* as given by Eq. (1.37), satisfies the Bellman equation

$$J(x) = \min_{u \in \Re} \big\{ qx^2 + ru^2 + J(ax + bu) \big\}, \tag{1.42}$$

and that $\mu^*(x)$, as given by Eqs. (1.40)-(1.41), 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 (1.42) is written as

$$\min_{u \in \Re} \big\{ qx^2 + ru^2 + K(ax + bu)^2 \big\}. \tag{1.43}$$

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. (1.43), we obtain

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

so the minimizing control and corresponding policy are given by

$$\mu_K(x) = L_K x, \tag{1.44}$$

where

$$L_K = -\frac{abK}{r + b^2 K}. \tag{1.45}$$

By substituting this control, the minimized expression (1.43) takes the form

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

After straightforward algebra, using Eq. (1.45) for L_K, it can be verified that this expression is written as $F(K)x^2$, with F given by Eq. (1.39). Thus when $J(x) = Kx^2$, the Bellman equation (1.42) takes the form

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

or equivalently $K = F(K)$ [cf. Eq. (1.38)].

In conclusion, when restricted to quadratic functions $J(x) = Kx^2$ with $K \geq 0$, the Bellman equation (1.42) is equivalent to the equation

$$K = F(K) = \frac{a^2 rK}{r + b^2 K} + q. \tag{1.46}$$

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. (1.44)-(1.45), attains the minimum in Bellman's equation, and is given by Eqs. (1.40)-(1.41).

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. (1.37)] 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

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

where K_L solves the equation

$$K = F_L(K), \tag{1.47}$$

† 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*, which transforms a function $J(x)$ into the right side of Bellman's equation,

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

also a function 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 monograph [Ber22a]). They will be introduced formally in Chapter 2, and they will be used to extend the analysis of the approximation in value space ideas of the present section.

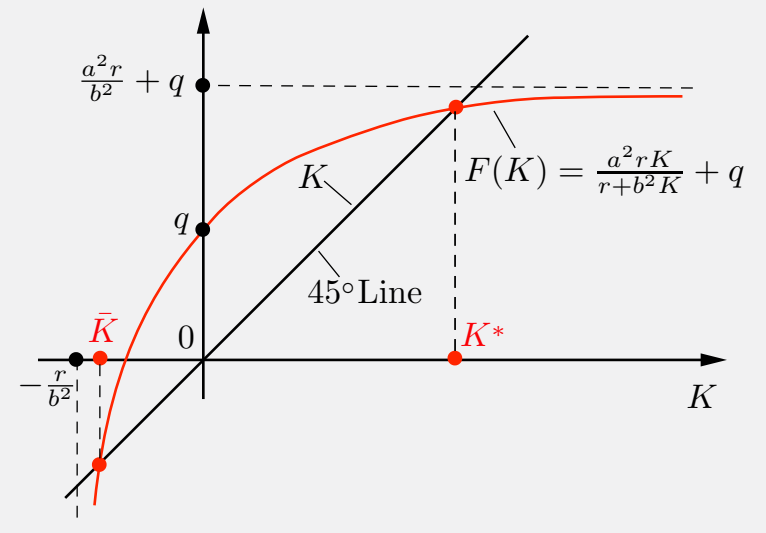

Figure 1.5.1 Graphical construction of the solutions of the Riccati equation (1.38)-(1.39) for the linear quadratic problem. The optimal cost function is $J^*(x) = K^*x^2$, where the scalar K^* solves the fixed point equation $K = F(K)$, with F being the function given by

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

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 lies within the negative real line and is of no interest.

with F_L defined by

$$F_L(K) = (a + bL)^2K + q + rL^2. \tag{1.48}$$

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 (1.47)-(1.48) for $\mu(x) = Lx$ may be justified by verifying that it is in fact the Bellman equation for μ,

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

[cf. Eq. (1.32)], restricted to quadratic functions of the form $J(x) = Kx^2$.

We note, however, that $J_\mu(x) = K_Lx^2$ is the solution of the Riccati equation (1.47)-(1.48) only when $\mu(x) = Lx$ is stable. If μ is unstable, i.e.,

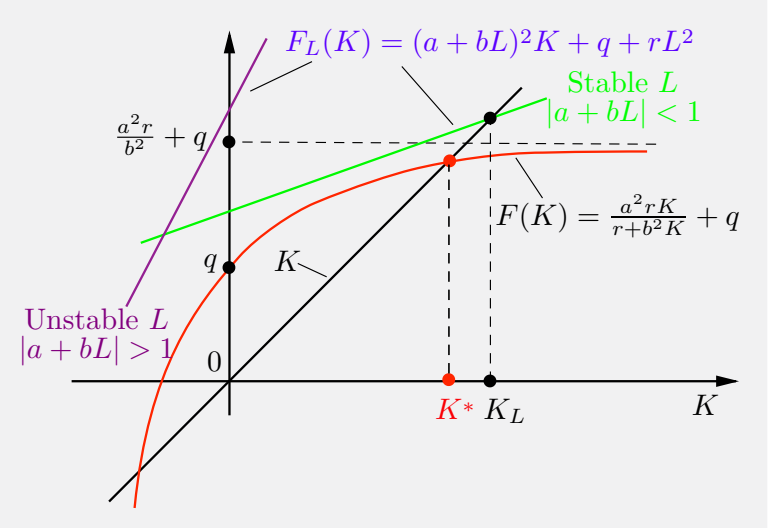

Figure 1.5.2 Illustration of the construction of the cost function of a linear policy $\mu(x) = Lx$, which is stable, i.e., $|a + bL| < 1$. The cost function $J_\mu(x)$ has the form

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

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 unstable with $|a + bL| > 1$, 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.

$|a + bL| \geq 1$, then (since $q > 0$ and $r > 0$) we have $J_\mu(x) = \infty$ for all $x \neq 0$. The Riccati equation (1.47)-(1.48) is still defined for an unstable policy, 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 $J_{k+1}(x) = K_{k+1}x^2$, where

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

with F being the Riccati operator of Eq. (1.46). The algorithm is illustrated in Fig. 1.5.3. As can be seen from the figure, when starting from any

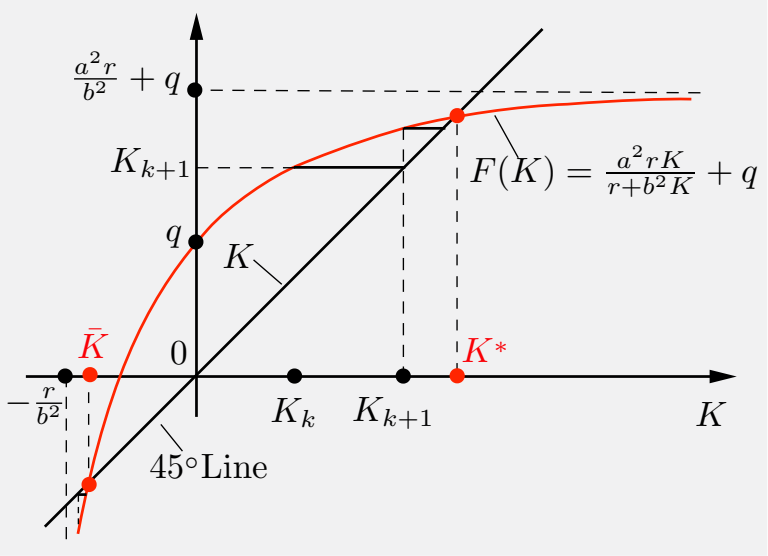

Figure 1.5.3 Graphical illustration of value iteration for the linear quadratic problem. It has the form $K_{k+1} = F(K_k)$, where F is the Riccati operator,

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

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

$K_0 \geq 0$, the algorithm generates a sequence $\{K_k\}$ of nonnegative scalars that converges to K^*.

1.5.1 Visualizing Approximation in Value Space - Newton's Method

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 that we will discuss in Chapter 2, in the context of more general infinite horizon problems. There we will use Bellman operators, which define the Bellman equations, in place of Riccati operators, which define the Riccati equations.

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. (1.44)-(1.45), by minimizing the right side of Bellman's equation when $J(x) = Kx^2$:

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

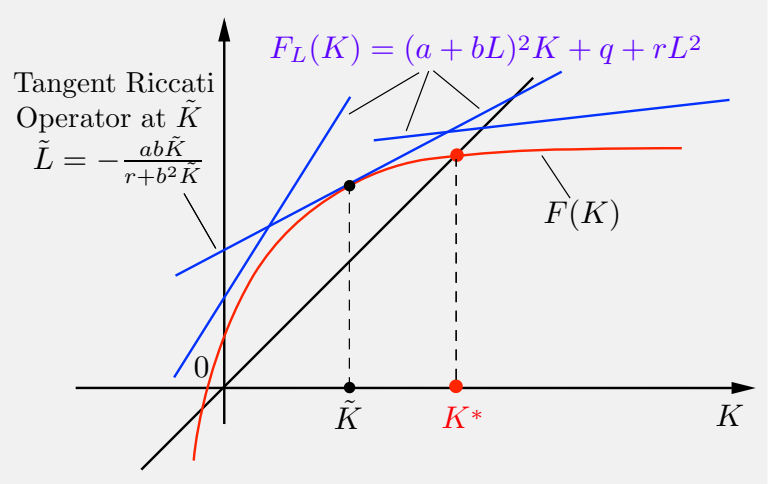

Figure 1.5.4 Illustration of how the graph of the Riccati operator F can be obtained as the lower envelope of the linear operators

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

as L ranges over the real numbers. We have $F(K) = \min_{L\in\Re} F_L(K)$, cf. Eq. (1.49). 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. (1.45)], 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.

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

$$F(K) = \min_{L\in\Re} F_L(K), \tag{1.49}$$

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

$$F_L(K) = (a + bL)^2K + q + bL. \tag{1.50}$$

Figure 1.5.4 illustrates the relation (1.49)-(1.50), 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, which we will denote by $\tilde{\mu}$. Figure 1.5.5 illustrates the corresponding linear function $F_{\tilde{L}}$, whose graph is a tangent line to the graph of F at the point K [cf. Fig. 1.5.4 and Eq. (1.50)].

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 $\tilde{K}$.*†

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 unstable, then in view of the positive definite quadratic cost per stage, we have $\tilde{\mu}(x) = \infty$ for all $x \neq 0$.

---

† 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. 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 system at the current iterate, but relax the differentiability requirement to piecewise differentiability, and/or component concavity, while maintaining the superlinear convergence property of the method; see the monograph [Ber22b] and the paper [Ber22c], which also provide a convergence analysis.

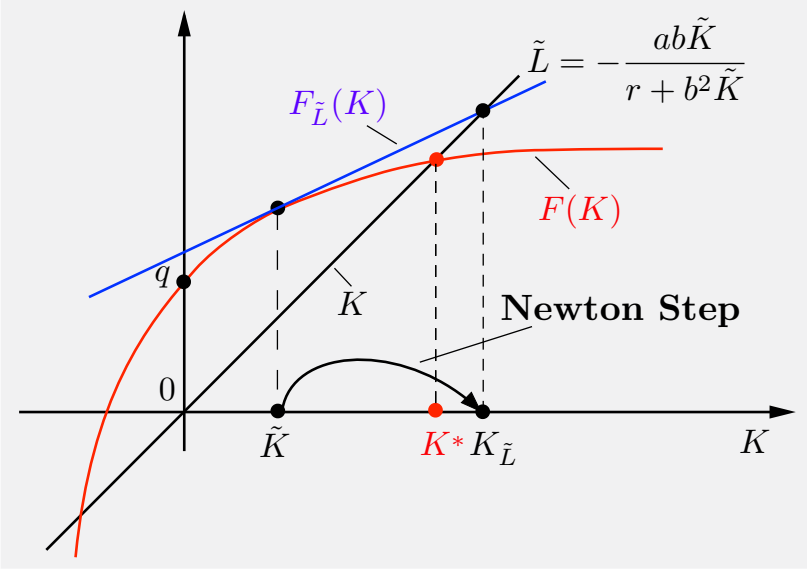

Figure 1.5.5 Illustration of approximation in value space with one-step lookahead for the linear quadratic problem. Given a terminal cost approximation $\tilde{J} = \tilde{K}x^2$, we compute the corresponding linear policy $\tilde{\mu}(x) = \tilde{L}x$, where

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

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

Multistep Lookahead

In the case of ℓ-step lookahead minimization, a similar Newton step interpretation 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 1.5.7 depicts the case $\ell = 2$.

It is interesting to note that as the length of lookahead increases, 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.*

Region of Stability

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

$$|a + bL_K| < 1.$$

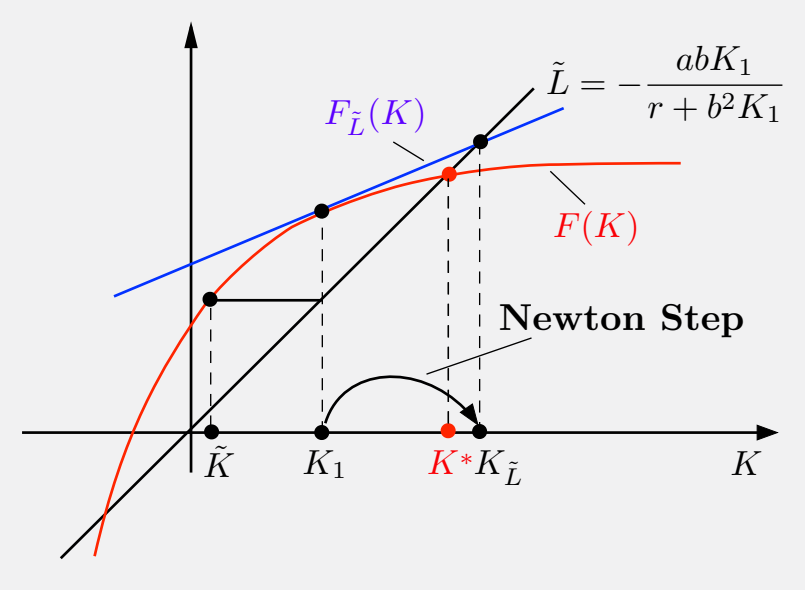

Figure 1.5.7 Illustration of approximation in value space with two-step lookahead for the linear quadratic problem. Starting with a terminal cost approximation $\tilde{J} = \tilde{K}x^2$, we obtain K_1 using a single value iteration. We then compute the corresponding linear policy $\tilde{\mu}(x) = \tilde{L}x$, 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.

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. 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. 1.5.6.

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*.

We summarize the Riccati equation formulas and the relation between linear policies of the form $\mu(x) = Lx$ and their quadratic cost functions in the following table.

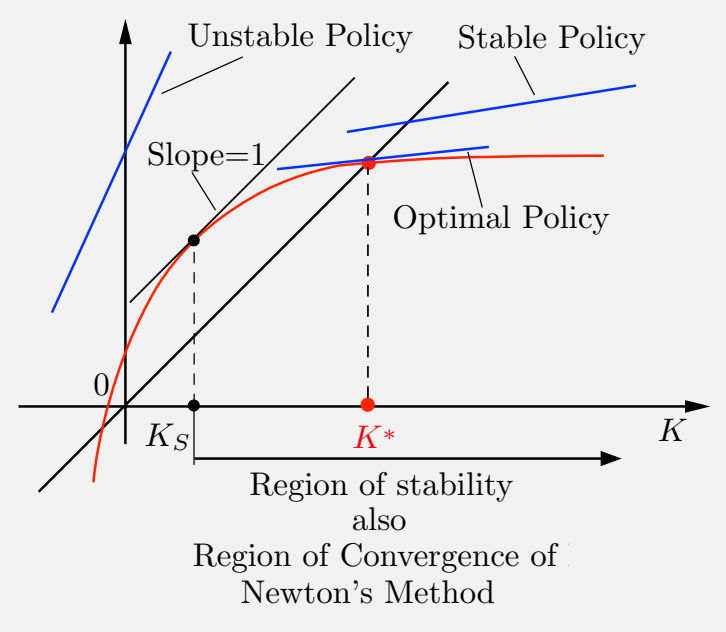

Figure 1.5.6 Illustration the region of stability, i.e., the set of $K \geq 0$ such that the one-step lookahead policy μ_K is stable. This is also the set of initial conditions for which Newton's method converges to K^* asymptotically.

Riccati Equation Formulas for One-Dimensional Problems

Riccati equation for minimization [cf. Eqs. (1.38) and (1.39)]

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

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. (1.45)]

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

Quadratic cost coefficient $\tilde{K}$ of a stable one-step lookahead policy
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.

1.5.2 Rollout and Policy Iteration

The rollout algorithm starts from some stable base policy μ, and obtains the rollout policy $\tilde{\mu}$ using a policy improvement operation, which by definition, yields the one-step lookahead policy that corresponds to terminal cost approximation J_μ. Figure 1.5.8 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 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 be linear and 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

$$J_{\mu^0}(x) = K_0 x^2, \tag{1.51}$$

where

$$K_0 = \frac{q + rL_0^2}{1 - (a + bL_0)^2}. \tag{1.52}$$

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

$$L_1 = -\frac{abK_0}{r + b^2K_0}; \tag{1.53}$$

cf. Eqs. (1.44)-(1.45).

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

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

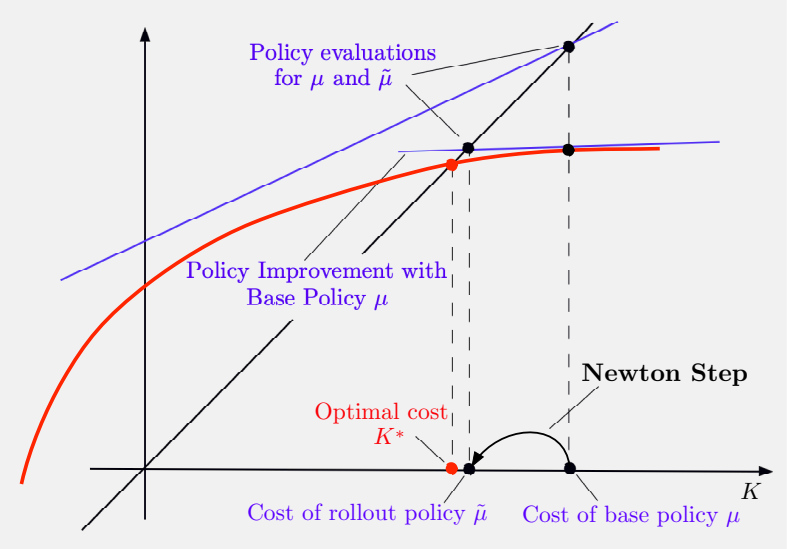

Figure 1.5.8 Illustration of the rollout algorithm for the linear quadratic problem. Starting from a linear stable base policy μ, it generates a stable rollout policy $\tilde{\mu}$. The quadratic cost coefficient of $\tilde{\mu}$ is obtained from the quadratic cost coefficient of μ with a Newton step for solving the Riccati equation.

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. (1.52)-(1.53)].

The corresponding cost function sequence has the form

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

cf. Eq. (1.51). 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 [Ber22b].

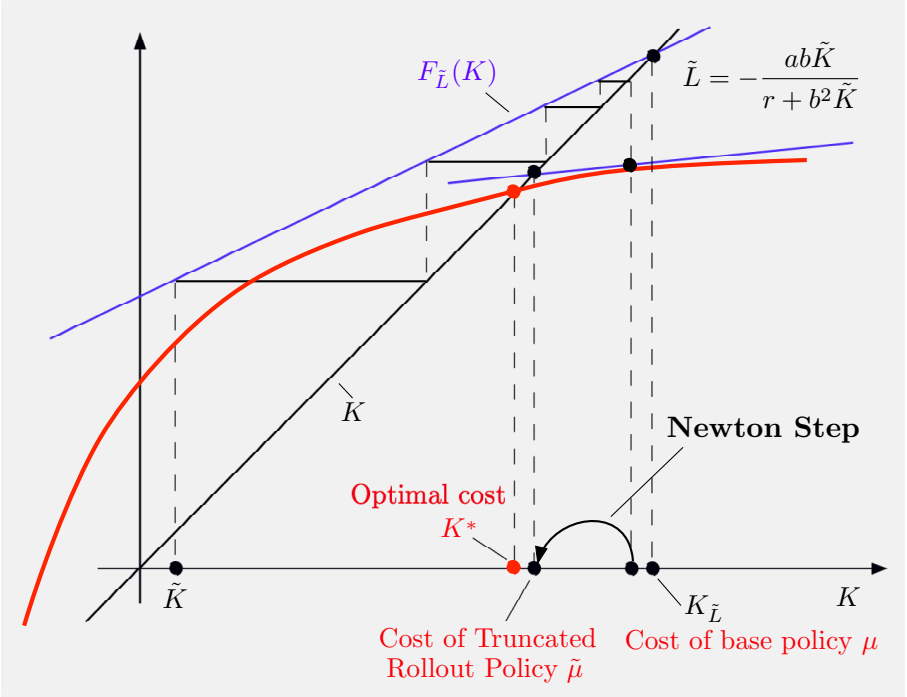

Figure 1.5.9 Illustration of truncated rollout with a stable base policy $\mu(x) = Lx$ and terminal cost approximation $\tilde{K}$ for the linear quadratic problem. In this figure, we use one-step lookahead minimization and the number of rollout steps is $m = 4$.

Truncated Rollout

An m-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. 1.5.9. 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, and we will now revisit these issues within the context of our linear quadratic problem. In particular we noted that:

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

(b) Lookahead by rollout may be an economic substitute for lookahead by minimization, in the sense that it may achieve a similar performance for the truncated rollout policy at significantly reduced computational

cost.

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. 1.5.9. We refer to the monograph [Ber22b] for further discussion.

1.6 EXAMPLES, VARIATIONS, AND SIMPLIFICATIONS

In this section we provide a few examples that illustrate problem formulation techniques, exact and approximate solution methods, and adaptations of the basic DP algorithm to various contexts. We refer to DP textbooks for extensive additional discussions of modeling and problem formulation techniques (see e.g, the many examples that can be found in the author's DP and RL textbooks [Ber12], [Ber17], [Ber19a], [Ber20a], as well as in the neuro-dynamic programming monograph [BeT96]).

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

1.6.1 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. Generally, a practical problem is a prime candidate for a DP formulation if it involves multiple sequential decisions separated by the collection of information that can enhance the effectiveness of future decisions.

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

duced artificially, with the aim to facilitate the use of approximate DP/RL methods.

There are also problems that fit quite well into the sequential structure of DP, but can be fruitfully addressed by RL methods that do not have a fundamental connection with DP. An important case in point is *policy gradient and policy search* methods, which will be considered in these notes, but will be discussed later in this class. 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 will be discussed further in Chapter 2. We also refer to Section 5.7 of the RL book [Ber19a] and to the end-of-chapter references.

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

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

(b) Select the states x_k. The basic guideline here is that x_k should encompass *all the information that is relevant for future optimization*, i.e., the information that is known to the controller at time k and can be used with advantage in choosing u_k. In effect, at time k *the state x_k should separate the past from the future*, in the sense that anything that has happened in the past (states, controls, and disturbances from stages prior to stage k) is irrelevant to the choices of future controls as long we know x_k. Sometimes this is described by saying that the state should have a "Markov property" to express an analogy with states of Markov chains, where (by definition) the conditional probability distribution of future states depends on the past history of the chain only through the present state.

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

Note that there may be multiple possibilities for selecting the states, because information may be packaged in several different ways that are equally useful from the point of view of control. It may thus be worth con-

sidering 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.

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.†

† 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.

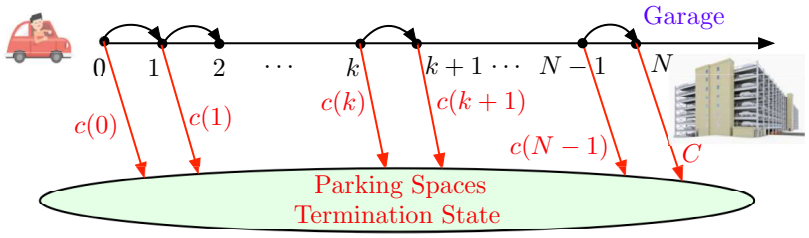

Figure 1.6.1 Cost structure of the parking problem. The driver may park at space $k = 0, 1, \ldots, N-1$ at cost $c(k)$, if the space is free, or continue to the next space $k+1$ at no cost. At space N (the garage) the driver must park at cost C.

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 policy, since it will not incur the large termination test 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 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

‡ When an upper bound on the number of stages to termination is not known, the problem may be formulated as an infinite horizon problem, a stochastic version of a shortest path problem.

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

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

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

Time Delays

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

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

$$x_{k+1} = f_k(x_k, x_{k-1}, u_k, u_{k-1}, w_k), \qquad k = 1, \ldots, N-1, \tag{1.54}$$

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

If we introduce additional state variables y_k and s_k, and we make the identifications $y_k = x_{k-1}$, $s_k = u_{k-1}$, the system equation (1.54) yields

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \\ s_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, y_k, u_k, s_k, w_k) \\ x_k \\ u_k \end{pmatrix}. \tag{1.55}$$

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

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

where the system function $\tilde{f}_k$ is defined from Eq. (1.55).

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

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

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

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

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

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

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

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

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

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

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

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

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

Forecasts

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

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

The forecasting process can be represented by means of the equation

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

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

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

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

The new state and disturbance are

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

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

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

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

$$J_k^*(x_k, y_k) = \min_{u_k \in U_k(x_k)} E_{w_k} \Big\{ g_k(x_k, u_k, w_k) + \sum_{i=1}^{m} p_i J_{k+1}^*\big(f_k(x_k, u_k, w_k), i\big) \mid y_k \Big\}, \tag{1.56}$$

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" as in the preceding example (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 preceding parking example, a DP algorithm that

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

$$\hat{J}_k(x_k) = E_{y_k}\Big\{ \min_{u_k \in U_k(x_k,y_k)} E_{w_k}\Big\{g_k(x_k,y_k,u_k,w_k) + \hat{J}_{k+1}\big(f_k(x_k,y_k,u_k,w_k)\big) \mid x_k,y_k,u_k\Big\} \Big| \, x_k\Big\}. \tag{1.57}$$

This is a consequence of the calculation

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

Note that the minimization in the right-hand side of the preceding equation must still be performed for all values of the full state (x_k, y_k) in order to yield an optimal control law as a function of (x_k, y_k). Nonetheless, the equivalent DP algorithm (1.57) 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. (1.57). 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. (1.56),

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

which is executed over the space of x_k rather than x_k *and* y_k. Note that this is a simpler algorithm to approximate than the one of Eq. (1.56).

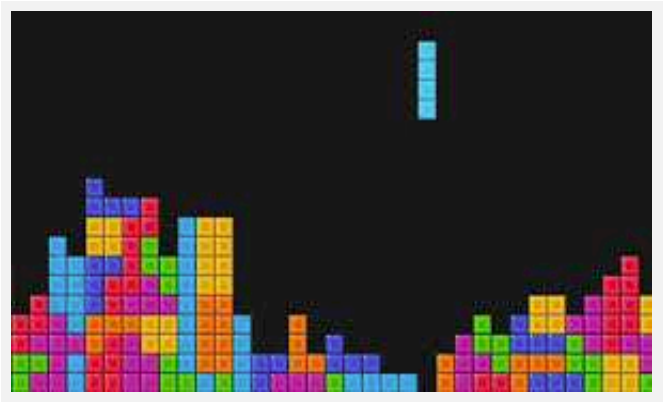

Figure 1.6.2 Illustration of a tetris board.

Uncontrollable state components often occur in arrival systems, such as queueing, where action must be taken in response to a random event (such as a customer arrival) that cannot be influenced by the choice of control. Then the state of the arrival system must be augmented to include the random event, but the DP algorithm can be executed over a smaller space, as per Eq. (1.57). Here is an example of this type.

Example 1.6.2 (Tetris)

Tetris is a popular video game played on a two-dimensional grid. Each square in the grid can be full or empty, making up a "wall of bricks" with "holes" and a "jagged top" (see Fig. 1.6.2). The squares fill up as blocks of different shapes fall from the top of the grid and are added to the top of the wall. As a given block falls, the player can move horizontally and rotate the block in all possible ways, subject to the constraints imposed by the sides of the grid and the top of the wall. The falling blocks are generated independently according to some probability distribution, defined over a finite set of standard shapes. The game starts with an empty grid and ends when a square in the top row becomes full and the top of the wall reaches the top of the grid. When a row of full squares is created, this row is removed, the bricks lying above this row move one row downward, and the player scores a point. The player's objective is to maximize the score attained (total number of rows removed) up to termination of the game, whichever occurs first.

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

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

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

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

The shape y is generated according to a probability distribution $p(y)$, independently of the control, so it can be viewed as an uncontrollable state component. The DP algorithm (1.57) is executed over the space of board positions x and has the intuitive form

$$\hat{J}_k(x) = \sum_y p(y) \max_u \Big[g(x,y,u) + \hat{J}_{k+1}\big(f(x,y,u)\big) \Big], \qquad \text{for all } x, \qquad (1.58)$$

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 (1.58) 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 (1.58), has the form

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

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

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

1.6.4 Partial State Information and Belief States

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

† Tetris has received a lot of attention as a challenging testbed for RL algorithms 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.

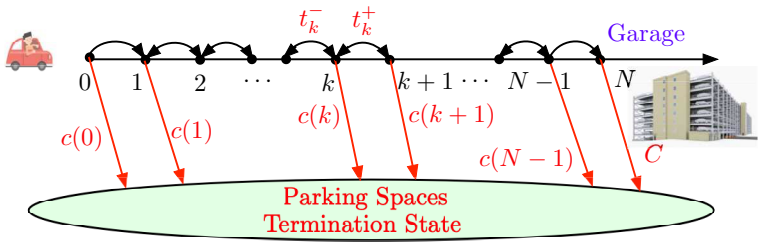

Figure 1.6.3 Cost structure and transitions of the bidirectional parking problem. The driver may park at space $k = 0, 1, \ldots, N-1$ at cost $c(k)$, if the space is free, can move to $k-1$ at cost t_k^- or can move to $k+1$ at cost t_k^+. At space N (the garage) the driver must park at cost C.

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., [Ber17], [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 in practice, even though there are DP algorithms for doing so. The most common approach is to replace the state x_k with a *belief state*, which we will denote by b_k. It is the probability distribution of x_k given all the observations that have been obtained by the controller up to time k, and it can serve as "state" in an appropriate DP algorithm. The belief state can in principle be computed and updated by a variety of methods that are based on Bayes' rule, such as *Kalman filtering* (see e.g., [AnM79], [KuV86], [Kri16], [ChC17]) and *particle filtering* (see e.g., [GSS93], [DoJ09], [Can16], [Kri16]).

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

Example 1.6.3 (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 status of the spaces previously visited and may return to any of these spaces; see Fig. 1.6.3.

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

Here the belief state is the vector of current probabilities

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

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

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

$$J_k^*(b_k) = \min_{u_k\in U_k}\left[\hat{g}_k(b_k,u_k) + E_{z_{k+1}}\Big\{J_{k+1}^*\big(F_k(b_k,u_k,z_{k+1})\big)\Big\}\right], \quad (1.59)$$

where:

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

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

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

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

$$b_{k+1} = F_k(b_k,u_k,z_{k+1}). \quad (1.60)$$

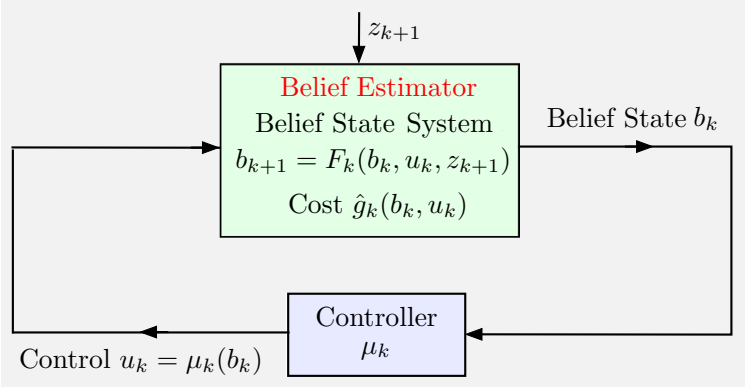

Figure 1.6.4 Schematic illustration of the view of an imperfect state information problem as one of perfect state information, whose state is the belief state b_k, i.e., the conditional probability distribution of x_k given all the observations up to time k. The observation z_{k+1} plays the role of the stochastic disturbance. The function F_k is a sequential estimator that updates the current belief state b_k.

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

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

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

in Eq. (1.59) is a function of b_k and u_k.

The algorithm (1.59) is just the ordinary DP algorithm for the perfect state information problem shown in Fig. 1.6.4. It involves the system/belief estimator (1.60) 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 (1.59) can only be executed approximately, using approximation in value space methods.

We refer to the textbook [Ber17], Chapter 4, for a detailed derivation of the DP algorithm (1.59), and to the monograph [BeS78] for a mathematical treatment that applies to infinite-dimensional state and disturbance spaces as well.

1.6.5 Multiagent Problems and Multiagent Rollout

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

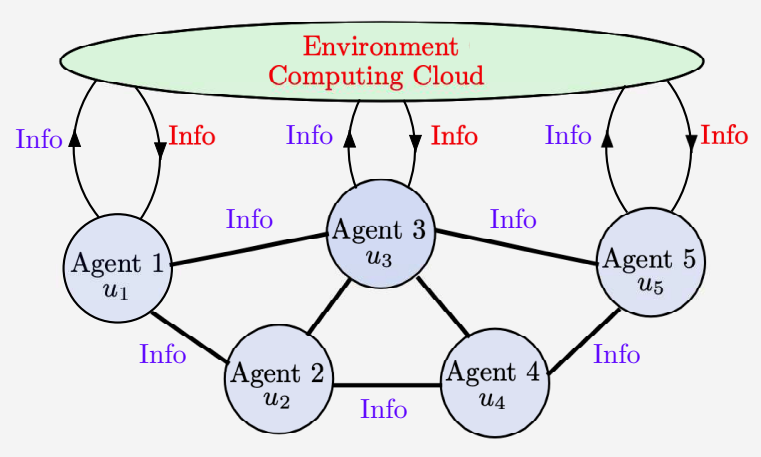

Figure 1.6.5 Schematic illustration of a multiagent problem. There are multiple "agents," and each agent $\ell = 1, \ldots, m$ controls its own decision variable u_ℓ. At each stage, agents exchange new information and also exchange information with the "environment," and then select their decision variables for the stage.

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

$$U(x) = U_1(x) \times \cdots \times U_m(x), \tag{1.61}$$

where the sets $U_\ell(x)$ are given. This structure is inspired by applications involving distributed decision making by multiple agents with communication and coordination between the agents; see Fig. 1.6.5.

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

$$\tilde{u} \in \arg\min_{u \in U(x)} E_w \Big\{ g(x,u,w) + \alpha J_\mu \big(f(x,u,w) \big) \Big\}, \tag{1.62}$$

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 (1.61)]. Thus the standard rollout algorithm requires an exponential [order $O(n^m)$] number of Q-factor computations per stage, which can be overwhelming even for moderate values of m.

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

minimization (1.62) is replaced by a sequence of m successive minimizations, *one-agent-at-a-time*, with the results incorporated into the subsequent minimizations. In particular, at state x we perform the sequence of minimizations

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

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

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

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

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

A key idea here is that the computational requirements of the rollout one-step minimization (1.62) are proportional to the number of controls in the set $U_k(x_k)$ and are independent of the size of the state space. This motivates a reformulation of the problem, first suggested in the book [BeT96], Section 6.1.4, whereby control space complexity is traded off with state space complexity, by "unfolding" the control u_k into its m components, which are applied *one agent-at-a-time* rather than all-agents-at-once.

In particular, we can reformulate the problem by breaking down the collective decision u_k into m 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 the generic state x, we break down the control u into the sequence of the m controls $u_1,u_2,\ldots,u_m$, and between x and the next state $\bar{x} = f(x,u,w)$, we introduce artificial intermediate "states" $(x,u_1),(x,u_1,u_2),\ldots,(x,u_1,\ldots,u_{m-1})$, and corresponding transitions. The choice of the last control component u_m at "state"

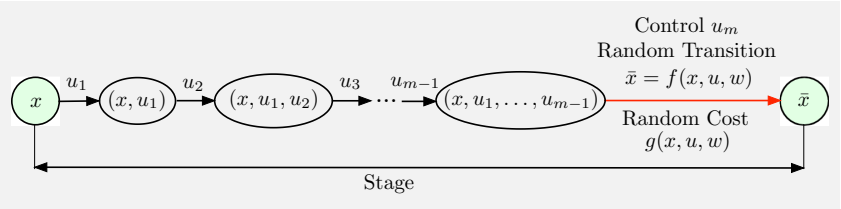

Figure 1.6.6 Equivalent formulation of the N-stage stochastic optimal control problem for the case where the control u consists of m components $u_1, u_2, \ldots, u_m$:

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

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

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

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

$(x, u_1, \ldots, u_{m-1})$ marks the transition to the next state $\bar{x} = f(x, u, w)$ according to the system equation, while incurring cost $g(x, u, w)$; see Fig. 1.6.6.

It is evident that this reformulated problem is equivalent to the original, since any control choice that is possible in one problem is also possible in the other problem, while the cost structure of the two problems is the same. In particular, every policy $\big(\mu_1(x), \ldots, \mu_m(x)\big)$ of the original problem, 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 (1.62) is performed for just one state at each stage.

The major fact that can be proved about multiagent rollout (see the end-of-chapter references) is that it still *achieves cost improvement*:

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

where $J_\mu(x)$ is the cost function of the base policy μ, and $J_{\tilde{\mu}}(x)$ is the cost function of the rollout policy $\tilde{\mu} = (\tilde{\mu}_1, \ldots, \tilde{\mu}_m)$, starting from state x. Furthermore, this cost improvement property can be extended to multiagent PI

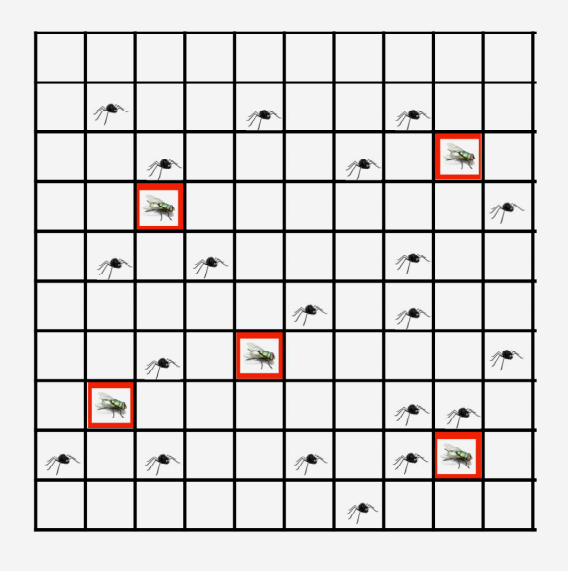

Figure 1.6.7 Illustration of a 2-dimensional spiders-and-fly problem with 20 spiders and 5 flies (cf. Example 1.6.4). The flies moves randomly, regardless of the position of the spiders. During a stage, each spider moves to a neighboring location or stays where it is, so there are 5 moves per spider (except for spiders at the edges of the grid). The total number of possible joint spiders moves is a little less than 5^{20}.

schemes that involve one-agent-at-a-time policy improvement operations, and have sound convergence properties. Moreover, multiagent rollout becomes the starting point for various related PI schemes that are well suited for distributed operation in important practical contexts involving multiple autonomous decision makers; see the book [Ber20a].

Example 1.6.4 (Spiders and Flies)

This example is representative of a broad range of practical problems such as multirobot service systems involving delivery, maintenance and repair, search and rescue, firefighting, etc. Here there are m spiders and several flies moving on a 2-dimensional grid; cf. Fig. 1.6.7. 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, 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.

1.6.6 Problems with Unknown Parameters - Adaptive and Model Predictive 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. Moreover, *problems with a known mathematical model are the only type that we will formally address in this book with DP and approximate DP methods*.

Of course in practice, it is common that the system parameters are either not known exactly or may change over time.† As an example consider

† The difficulties of decision and control within a changing environment are often underestimated. Among others, they complicate the balance between off-

our oversimplified cruise control system of Example 1.3.1 or its infinite horizon version. The state evolves according to

$$x_{k+1} = x_k + bu_k + w_k, \tag{1.63}$$

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 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. This is one of the most challenging areas of stochastic optimal and suboptimal control, and has been studied 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

---

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.

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. (1.63) 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. We will not discuss adaptive control in this book, except peripherally. 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.

Dealing with Unknown Parameters by 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. 1.6.8.

One drawback of this approach is that it is not always easy to determine when to terminate one phase and start the other. A second difficulty,

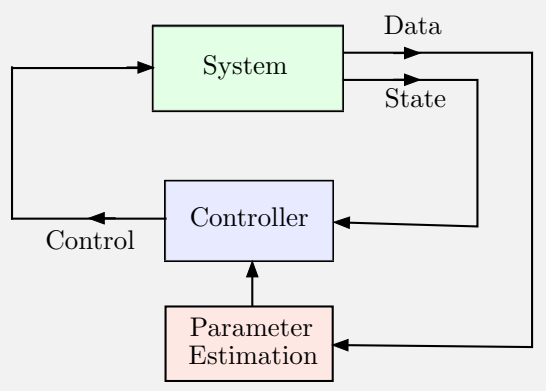

Figure 1.6.8 Schematic illustration of concurrent parameter estimation and system control. The system parameters are estimated on-line and the estimates are periodically passed on to the controller.

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 [Ber17], Section 6.7. 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

$$\tilde{\mu}_k(x_k) = \gamma x_k \tag{1.64}$$

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 (1.64), 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 (1.64) 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. 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 1.6.4).

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,

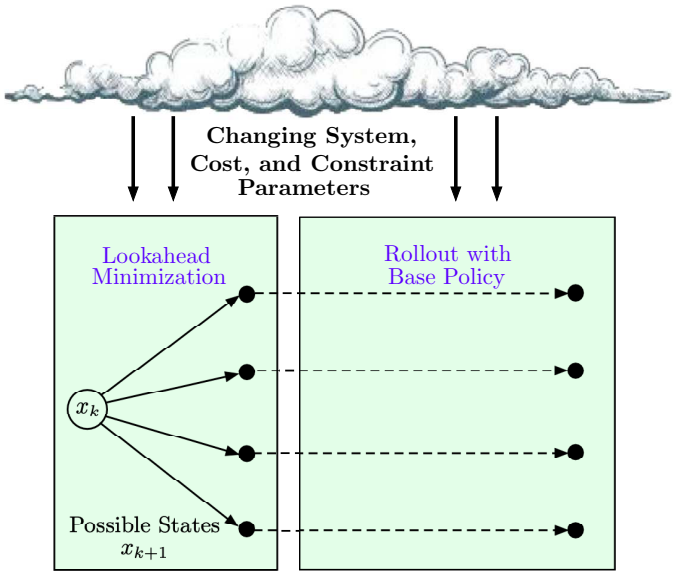

Figure 1.6.9 Schematic illustration of adaptive control by rollout. One-step lookahead is followed by simulation with the base policy, which stays fixed. The system, cost, and constraint parameters are changing over time, and the most recent values are incorporated into the lookahead minimization and 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.

but rollout with the current policy used as the base policy is particularly attractive. Here 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. 1.6.9). Note also 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 changing parameters.

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 1.6.5 (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

$$K = \frac{rK}{r + b^2 K} + 1. \tag{1.65}$$

The optimal policy has the form

$$\mu^*(x) = L^* x, \tag{1.66}$$

where

$$L^* = -\frac{bK^*}{r + b^2 K^*}. \tag{1.67}$$

As an example, consider the optimal policy that corresponds to the nominal problem parameters $b = 2$ and $r = 0.5$: this is the policy (1.66)-(1.67), with K obtained as the positive solution of the quadratic Riccati Eq. (1.65) for $b = 2$ and $r = 0.5$. We thus obtain

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

From Eq. (1.67) we then have

$$L = -\frac{2+\sqrt{6}}{5+2\sqrt{6}}. \tag{1.68}$$

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. (1.65).

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

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

where L is given by Eq. (1.68). 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 1.6.10 shows the coefficients K^*, K_L, and $\tilde{K}_L$ for a range of values of r and b. We have

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

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

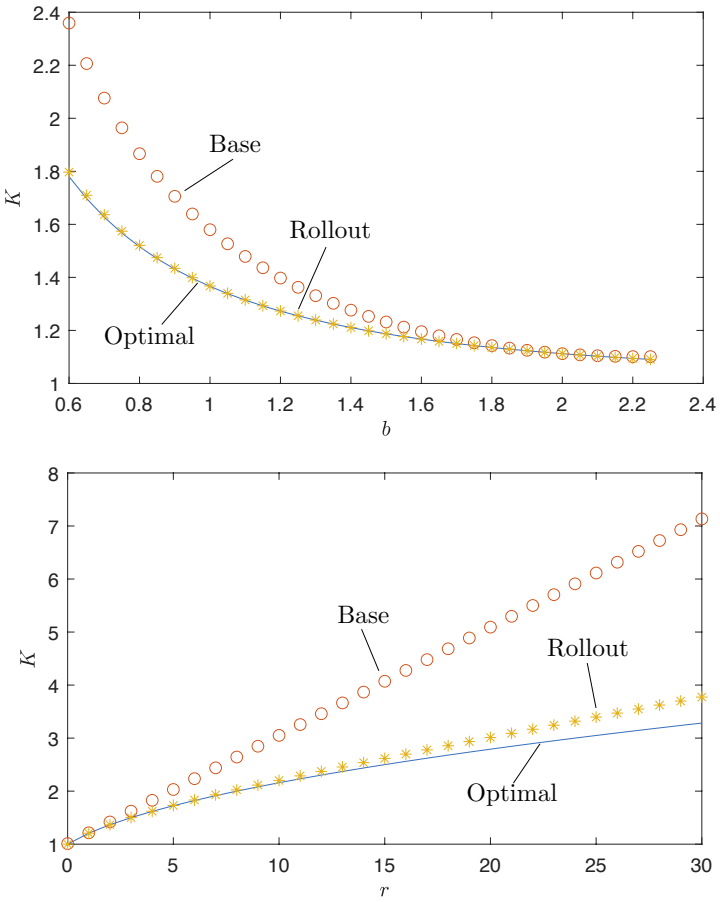

Figure 1.6.10 Illustration of control by rollout under changing problem parameters. The quadratic cost coefficients K^* (optimal, denoted by solid line), K_L (base policy, denoted by circles), and $\tilde{K}_L$ (rollout policy, denoted by asterisks) for the two cases where $r = 0.5$ and b varies, and $b = 2$ and r varies. The value of L is fixed at the value that is optimal for $b = 2$ and $r = 0.5$ [cf. Eq. (1.68)]. The rollout policy is very close to optimal, even when the base policy is far from optimal. This is related to the size of the second derivative of the Riccati Eq. (1.65); see Exercise 1.3.

Note that, as the figure illustrates, we have

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

where for a given initial state, $\tilde{J}$ is the rollout performance, J^* is the optimal performance, and J is the base policy performance. This is a consequence of the superlinear/quadratic convergence rate of Newton's method that underlies rollout, and guarantees that the rollout performance approaches the optimal much faster than the base policy performance does.

Note that Fig. 1.6.10 illustrates the behavior of the error ratio

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

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

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

Example 1.6.6 (Model Predictive Control, Rollout, and On-Line Replanning)

Let us briefly discuss the MPC methodology, with a view towards its connection with the rollout algorithm. Consider an undiscounted infinite horizon deterministic problem, involving the system

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

whose state x_k and control u_k are finite-dimensional vectors. The cost per stage is assumed nonnegative

$$g(x_k, u_k) \geq 0, \qquad \text{for all } (x_k, u_k),$$

(e.g., a quadratic cost). There are control constraints $u_k \in U(x_k)$, and to simplify the following discussion, we will assume that there are no state constraints. We assume that the system can be kept at the origin at zero cost, i.e.,

$$f(0, \overline{u}_k) = 0, \quad g(0, \overline{u}_k) = 0 \qquad \text{for some control } \overline{u}_k \in U(0).$$

For a given initial state x_0, we want to obtain a sequence $\{u_0, u_1, \ldots\}$ that satisfies the control constraints, while minimizing the total cost. This is a classical problem in control system design, where the aim is to keep the state of the system near the origin (or more generally some desired set point), in the face of disturbances and/or parameter changes.

The MPC algorithm at each encountered state x_k applies a control that is computed as follows; see Fig. 1.6.11:

(a) It solves an ℓ-stage optimal control problem involving the same cost function and the requirement $x_{k+\ell} = 0$. This is the problem

$$\min_{u_t,\, t=k,\ldots,k+\ell-1} \sum_{t=k}^{k+\ell-1} g(x_t, u_t), \tag{1.69}$$

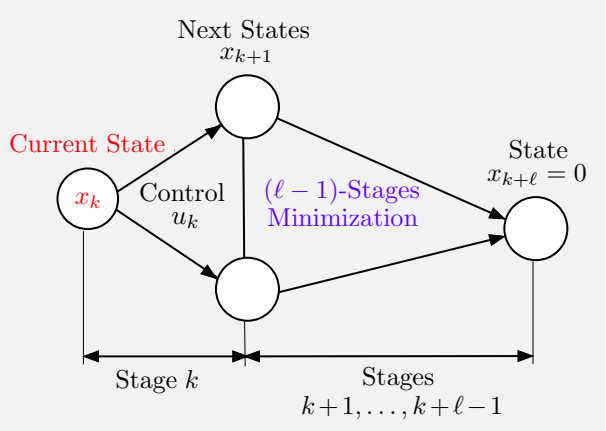

Figure 1.6.11 Illustration of the problem solved by MPC at state x_k. We minimize the cost function over the next ℓ stages while imposing the requirement that $x_{k+\ell}=0$. We then apply the first control of the optimizing sequence. In the context of rollout, the minimization over u_k is the one-step lookahead, while the minimization over $u_{k+1},\ldots,u_{k+\ell-1}$ that drives $x_{k+\ell}$ to 0 is the base heuristic.

subject to the system equation constraints

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

the control constraints

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

and the terminal state constraint $x_{k+\ell}=0$. Here ℓ is an integer with $\ell>1$, which is chosen in some largely empirical way.

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

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

To make the connection of MPC with rollout, we note that *the one-step lookahead function $\tilde{J}$ implicitly used by MPC [cf. Eq. (1.69)] is the cost-to-go function of a certain base heuristic*. This is the heuristic that drives to 0 the state after $\ell-1$ stages (*not ℓ stages*) and keeps the state at 0 thereafter, while observing the state and control constraints, and minimizing the associated $(\ell-1)$-stages cost. This rollout view of MPC, first discussed in the author's paper [Ber05a], is useful for making a connection with the approximate DP/RL techniques that are the main subject of this book, such as truncated rollout, cost function approximations, PI, etc.

Returning to the issue of dealing with changing problem parameters, it is natural to consider on-line replanning as per our earlier discussion. In particular, once new estimates of system and/or cost function parameters

become available, MPC can adapt accordingly by introducing the new parameter estimates into the ℓ-stage optimization problem in (a) above.

Let us also note a common variant of MPC, where the requirement of driving the system state to 0 in ℓ steps in the ℓ-stage MPC problem (1.69), is replaced by a terminal cost $G(x_{k+\ell})$. Thus at state x_k, we solve the problem

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

instead of problem (1.69) where we require that $x_{k+\ell} = 0$. This variant can also be viewed as rollout with one-step lookahead, and a base heuristic, which at state x_{k+1} applies the first control $\tilde{u}_{k+1}$ of the sequence $\{\tilde{u}_{k+1}, \ldots, \tilde{u}_{k+\ell-1}\}$ that minimizes

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

Note that the performance of the MPC controller may be much better than the performance of this base heuristic (in relative terms) if the base heuristic is close to optimal [which is true if $G(x_{k+\ell}) \approx J^*(x_{k+\ell})$]. This is because in view of the superlinear/quadratic convergence rate of Newton's method that underlies rollout, we have

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

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

The on-line replanning scheme suffers from a potential computational bottleneck, associated with fast estimation of new system parameter values and the associated policy recalculation. A variation of on-line replanning, aims to deal with this difficulty by precomputation. In particular, assume that the set of problem parameters may take a known finite set of values.† 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.

1.7 REINFORCEMENT LEARNING AND OPTIMAL CONTROL - SOME TERMINOLOGY

The current state of RL has greatly benefited from the cross-fertilization of ideas from optimal control and from artificial intelligence. The strong connections between these two fields are now widely recognized. Still, however,

† 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.

substantial differences in language and emphasis remain between RL-based discussions (where artificial intelligence-related terminology is used) and DP-based discussions (where optimal control-related terminology is used).

The terminology used in this book is standard in DP and optimal control, and in an effort to forestall confusion of readers that are accustomed to either the artificial intelligence or the optimal control terminology, we provide a list of terms commonly used in RL, and their optimal control counterparts.

(a) **Environment** = System.

(b) **Agent** = Decision maker or controller.

(c) **Action** = Decision or control.

(d) **Reward of a stage** = (Opposite of) Cost of a stage.

(e) **State value** = (Opposite of) Cost starting from a state.

(f) **Value (or reward) function** = (Opposite of) Cost function.

(g) **Maximizing the value function** = Minimizing the cost function.

(h) **Action (or state-action) value** = Q-factor (or Q-value) of a state-control pair. (Q-value is also used often in RL.)

(i) **Planning** = Solving a DP problem with a known mathematical model.

(j) **Learning** = Solving a DP problem without using an explicit mathematical model. (This is the principal meaning of the term "learning" in RL. Other meanings are also common.)

(k) **Self-learning** (or self-play in the context of games) = Solving a DP problem using some form of policy iteration.

(l) **Deep reinforcement learning** = Approximate DP using value and/or policy approximation with deep neural networks.

(m) **Prediction** = Policy evaluation.

(n) **Generalized policy iteration** = Optimistic policy iteration.

(o) **State abstraction** = State aggregation.

(p) **Temporal abstraction** = Time aggregation.

(q) **Learning a model** = System identification.

(r) **Episodic task or episode** = Finite-step system trajectory.

(s) **Continuing task** = Infinite-step system trajectory.

(t) **Experience replay** = Reuse of samples in a simulation process.

(u) **Bellman operator** = DP mapping or operator.

(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.

Notation

Unfortunately the confusion arising from different terminology has been exacerbated by the use of different notations. The present book roughly follows the "standard" notation of the Bellman/Pontryagin optimal control era; see e.g., the classical books by Athans and Falb [AtF66], Bellman [Bel67], and Bryson and Ho [BrH75]. This notation is 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 our most prominently used symbols is as follows:

(a) x: state.

(b) u: control.

(c) J: cost function.

(d) g: cost per stage.

(e) f: system function.

(f) i: discrete state.

(g) $p_{xy}(u)$: transition probability from state x to state y under control u.

(h) α: discount factor in discounted problems.

The x-u-J notation is standard in optimal control textbooks (e.g., the books by Athans and Falb [AtF66], and Bryson and Ho [BrH75], as well as the more recent book by Liberzon [Lib11]). The notations f and g are also used most commonly in the literature of the early optimal control period as well as later (unfortunately the more natural symbol "c" has not been used much in place of "g" for the cost per stage). The discrete system notations i and $p_{ij}(u)$ are very common in the discrete-state Markov decision problem and operations research literature, where discrete-state problems have been treated extensively [sometimes the alternative notation $p(j \mid i, u)$ is used for the transition probabilities].

The artificial intelligence literature addresses for the most part finite-state Markov decision problems, most frequently the discounted and sto-

chastic shortest path infinite horizon problems that are discussed in Chapter 5. The most commonly used notation is s for state, a for action, $r(s, a, s')$ for reward per stage, $p(s' \mid s, a)$ or $p(s, a, s')$ for transition probability from s to s' under action a, and γ for discount factor. However, this type of notation is not well suited for continuous spaces models, which are of major interest for this book. The reason is that it requires the use of transition probability distributions defined over continuous spaces, and it leads to more complex and less intuitive mathematics. Moreover the transition probability notation makes no sense for deterministic problems, which involve no probabilistic structure at all.

1.8 NOTES, SOURCES, AND EXERCISES

Sections 1.1-1.4: Our discussion of exact DP in this chapter has been brief since our focus in this course will be on approximate DP and RL. The author's DP textbook [Ber17] provides an extensive discussion of finite horizon exact DP, and its applications to discrete and continuous spaces problems, using a notation and style that is consistent with the one used here. The books by Puterman [Put94] and by the author [Ber12] provide detailed treatments of infinite horizon finite-state stochastic DP problems. The book [Ber12] also covers continuous/infinite state and control spaces problems, including the linear quadratic problems that we have discussed in this chapter through examples. Continuous spaces problems present special analytical and computational challenges, which are at the forefront of research of the RL methodology.

Some of the more complex mathematical aspects of exact DP are discussed in the monograph by Bertsekas and Shreve [BeS78], particularly the probabilistic/measure-theoretic issues associated with stochastic optimal control, including partial state information problems. The second volume of the author's DP book [Ber12] provides in an appendix an accessible summary introduction of the measure-theoretic framework of the book [BeS78], while a long paper by Yu and Bertsekas [YuB15] contains recent supplementary research with a particular focus on the analysis of policy iteration methods, which were not treated in [BeS78].

The author's abstract DP monograph [Ber18a], [Ber22a] aims at a unified development of the core theory and algorithms of total cost sequential decision problems, and addresses simultaneously stochastic, minimax, game, risk-sensitive, and other DP problems, through the use of abstract DP operators (or Bellman operators as we call them here). The idea here is to gain insight through abstraction. In particular, the structure of a DP model is encoded in its abstract Bellman operator, which serves as the "mathematical signature" of the model. Thus, characteristics of this operator (such as monotonicity and contraction) largely determine the analytical results and computational algorithms that can be applied to that model.

Abstract DP ideas are also useful for visualizations and interpretations of RL methods using the Newton method formalism that we discuss in these notes.

Rollout algorithms for discrete optimization problems were introduced in the paper by Bertsekas, Tsitsiklis, and Wu [BTW97], and the neuro-dynamic programming book [BeT96]. They are described in varying levels of detail in the RL books [Ber19a], [Ber20a], which also include extensive journal references to successful applications.

Approximation in value space, rollout, policy iteration, and the associated Newton method formalism are the principal subjects of these notes.† These are very powerful and general techniques: they can be applied to deterministic and stochastic problems, finite and infinite horizon problems, discrete and continuous spaces problems, and mixtures thereof. Rollout is reliable, easy to implement, and can be used in conjunction with on-line replanning.

As we have noted, rollout with a given base policy is simply the first iteration of the policy iteration algorithm starting from the base policy. Truncated rollout will be interpreted later in this course as an "optimistic" form of a single policy iteration, whereby a policy is evaluated inexactly, by using a limited number of value iterations.‡

† The name "rollout" (also called "policy rollout") was introduced by Tesauro and Galperin [TeG96] in the context of rolling the dice in the game of backgammon. In Tesauro's proposal, a given backgammon position is evaluated by "rolling out" many games starting from that position to the end of the game. To quote from the paper [TeG96]: "In backgammon parlance, the expected value of a position is known as the "equity" of the position, and estimating the equity by Monte-Carlo sampling is known as performing a "rollout." This involves playing the position out to completion many times with different random dice sequences, using a fixed policy P to make move decisions for both sides."

‡ Truncated rollout was also proposed in [TeG96]. To quote from this paper: "Using large multi-layer networks to do full rollouts is not feasible for real-time move decisions, since the large networks are at least a factor of 100 slower than the linear evaluators described previously. We have therefore investigated an alternative Monte-Carlo algorithm, using so-called "truncated rollouts." In this technique trials are not played out to completion, but instead only a few steps in the simulation are taken, and the neural net's equity estimate of the final position reached is used instead of the actual outcome. The truncated rollout algorithm requires much less CPU time, due to two factors: First, there are potentially many 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

Policy iteration, which will be viewed here as the repeated use of rollout, is more ambitious and challenging than rollout. It requires off-line training, possibly in conjunction with the use of neural networks. Together with its neural network and distributed implementations, it will be discussed in more detail later.

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 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 connection of approximation in value space with Newton's method, and its connections with MPC and adaptive control was first presented in the author's papers [Ber21], [Ber22c], and monograph [Ber22b]. This connection is the starting point for the new research presented in this course.

Section 1.6: Many applications of DP are presented in the 1st volume of the author's DP book [Ber17]. This book also covers a broad variety of state augmentation and problem reformulation techniques, including the mathematics of how problems with imperfect state information (POMDP) can be transformed to perfect state information problems.

Multiagent problem research 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 for the field at that time were *team theory* and *decentralized control*. For a sampling of subsequent works in team theory and multiagent optimization, we refer to the papers by Krainak, Speyer, and Marcus [KLM82a], [KLM82b], and de Waal and van Schuppen [WaS00]. For more recent works, see Nayyar, Mahajan, and Teneketzis [NMT13], Nayyar and Teneketzis [NaT19], Li et al. [LTZ19], Qu and Li [QuL19], Gupta [Gup20], the book by Zoppoli, Sanguineti, Gnecco, and Parisini [ZSG20], and the references quoted there. 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].

We note that the term "multiagent" has been used with several different meanings in the literature. For example some authors place emphasis on the case where the agents do not have common information when selecting their decisions. This gives rise to sequential decision problems with "nonclassical information patterns," which can be very complex, partly because they cannot be addressed by exact DP. Other authors adopt as their starting point a problem where the agents are "weakly" coupled through

---

assessment.

the system equation, the cost function, or the constraints, and consider methods whereby the weak coupling is exploited to address the problem through (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], and the papers [Ber19d], [Ber20b]. A computational study where several of the multiagent algorithmic ideas were tested and validated is the paper by Bhattacharya et al. [BKB20]. This paper considers a large-scale multi-robot routing and repair problem, involving partial state information, and explores some of the attendant implementation issues, including autonomous multiagent rollout, through the use of policy neural networks and other precomputed signaling policies.

The subject of adaptive control has a long history and its literature is very extensive; see the books by Aström and Wittenmark [AsW94], Aström and Hagglund [AsH95], [AsH06], 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], Bodson [Bod20]. 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 (also known as the "dual control problem"), to ARMAX/time series models, to extremum-seeking methods, to simulation-based RL techniques, etc.†

The research on problems involving unknown models and using data for model identification prior to or simultaneously with control was rekindled with the advent of the artificial intelligence side of RL and its focus on the active exploration of the environment. Here there is emphasis in "learning from interaction with the environment" [SuB18] through the use of (possibly hidden) Markov decision models, machine learning, and neural networks, in a wide array of methods that are under active development at present. 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. In these notes we will not deal with control of unknown models, except tangentially, and in the context of on-line replanning for problems with changing model parameters.

† The ideas of PID control originated even earlier. According to Wikipedia, "a formal control law for what we now call PID or three-term control was first developed using theoretical analysis, by Russian American engineer Nicolas Minorsky" [Min22].

The literature on 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]. The connections between MPC and rollout were discussed in the author's survey [Ber05a]. Textbooks on MPC include Maciejowski [Mac02], Goodwin, Seron, and De Dona [GSD06], Camacho and Bordons [CaB07], Kouvaritakis and Cannon [KoC16], Borrelli, Bemporad, and Morari [BBM17], and Rawlings, Mayne, and Diehl [RMD17].

Reinforcement Learning Sources

The approximate DP and RL literature has expanded tremendously since the connections between DP and RL became apparent in the late 80s and early 90s. We restrict ourselves to mentioning textbooks, research monographs, and broad surveys, which supplement our discussions, express related viewpoints, and collectively provide a guide to the literature. Moreover, inevitably our referencing reflects a cultural bias, and an overemphasis on sources that are familiar to the author and are written in a similar style to the present book (including the author's own works). Thus we wish to apologize in advance for the many omissions of important research references that are somewhat outside our own understanding and view of the field.

Two books were written in the 1990s, setting the tone for subsequent developments in the field. One in 1996 by Bertsekas and Tsitsiklis [BeT96], which reflects a decision, control, and optimization viewpoint, and another in 1998 by Sutton and Barto, which reflects an artificial intelligence viewpoint (a 2nd edition, [SuB18], was published in 2018). We refer to the former book and also to the author's DP textbooks [Ber12], [Ber17] 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 also of major interest in this book.

More recent books are by Gosavi [Gos15] (a much expanded 2nd edition of his 2003 monograph), which emphasizes simulation-based optimization and RL algorithms, Cao [Cao07], which focuses on a sensitivity approach to simulation-based 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.

There are also several books that, while not exclusively focused on DP and/or RL, touch upon several of the topics of this book. The book by Borkar [Bor08] is an advanced monograph that addresses rigorously many of the convergence issues of iterative stochastic algorithms in approximate DP, mainly using the so called ODE approach. The book by Meyn [Mey07] is broader in its coverage, but discusses some of the popular approximate DP/RL algorithms. The book by Haykin [Hay08] discusses approximate DP in the broader context of neural network-related subjects. The book by Krishnamurthy [Kri16] focuses on partial state information problems, with discussion of both exact DP, and approximate DP/RL methods. The 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 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 book by Lattimore and Szepesvari [LaS20] is focused on multiarmed bandit methods.

The present notes are similar in style, terminology, and notation to the author's recent RL textbooks [Ber19a], [Ber20a], [Ber22b], and the 3rd edition of the abstract DP monograph [Ber22a], which collectively provide a fairly comprehensive account of the subject. In particular, the 2019 RL textbook includes a broader coverage of approximation in value space methods, including certainty equivalent control and aggregation methods. It also covers substantially policy gradient methods for approximation in policy space, which we will not address here. The 2020 book focuses more closely on rollout, policy iteration, and multiagent problems. The 2022 book focuses on the connection of approximation in value space with Newton's method, relying on analysis first provided in the book [Ber20a] and

the paper [Ber22c]. The abstract DP monograph [Ber22a] is an advanced treatment of exact DP, which however connects with some of the visualizations used in the present notes.

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 several 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), 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), the author's [Ber05a] (which focuses on rollout algorithms and model predictive control), [Ber11a] (which focuses on approximate policy iteration), and [Ber18b] (which focuses on aggregation methods), and Recht [Rec18a] (which focuses on continuous spaces optimal control).

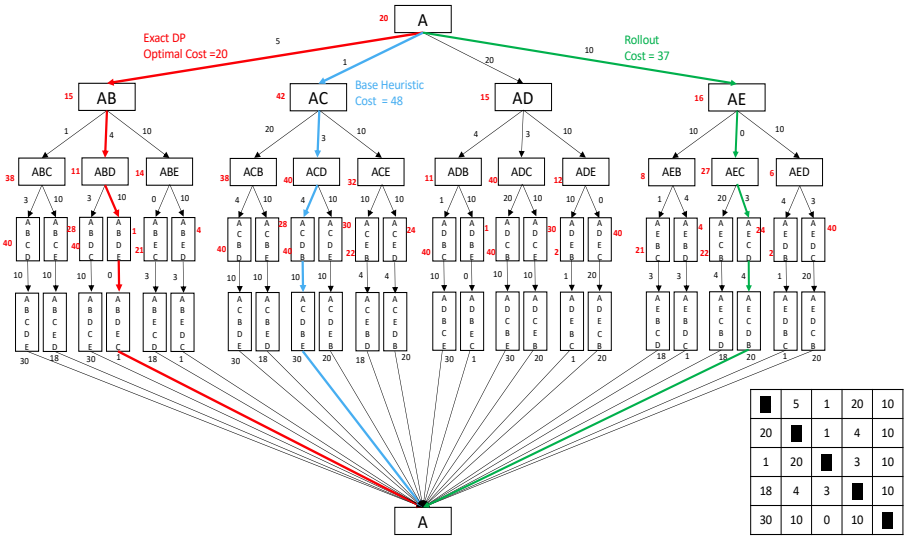

Figure 1.8.1 Solution of parts (a), (b), and (c) of Exercise 1.1. A 5-city traveling salesman problem illustration of rollout with the nearest neighbor base heuristic.

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 42 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 - A Stochastic Investment Problem)

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 the following problem.

An investor wants 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 is 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

$$0 < p^- < p^+, \qquad p^+ + p^- < 1.$$

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) 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:

$$J_N^*(x_N) = x_N, \tag{1.70}$$

and for $k = 0, \ldots, N-1$, if $x_k = 0$, then

$$J_k^*(0) = p^+ J_{k+1}^*(1) + (1 - p^+) J_{k+1}^*(0), \tag{1.71}$$

if $x_k = \bar{x}$, then

$$J_k^*(\bar{x}) = \bar{x}, \tag{1.72}$$

(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

$$J_k^*(x_k) = \max\left\{x_k,\, p^+ J_{k+1}^*(x_k+1) + (1-p^+-p^-) J_{k+1}^*(x_k) + p^- J_{k+1}^*(x_k-1)\right\}. \tag{1.73}$$

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 in Fig. 1.8.2. All the calculations are done for the following special case:

$$N = 12, \quad x_0 = 2, \quad \bar{x} = 10, \quad p^+ = p^- = 0.25.$$

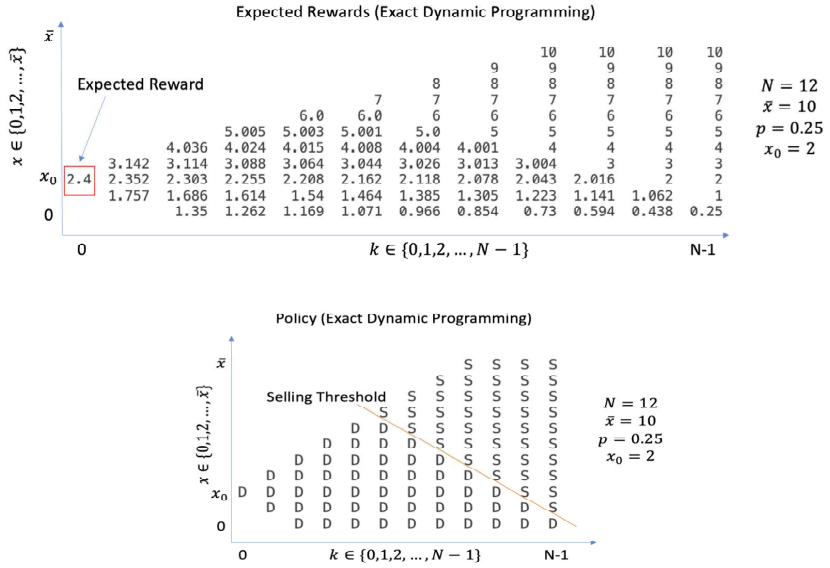

Figure 1.8.2 Table of values of optimal reward-to-go, obtained by exact DP, and corresponding optimal policy [cf. the algorithm (1.70)-(1.73). Only the states x_k that are reachable from x_0 at time k are considered (this is the state space for time k).

These values are also used for parts (b) and (c). 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

$$N = 10, \quad x_0 = 20, \quad \bar{x} = 40, \quad p^+ = p^- = 0.25.$$

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 policy, 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 policy 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 policy is not sequentially consistent.) The algorithm is given by

$$J_N^{x_k}(x_N) = x_N, \tag{1.74}$$

and for $n = k, \dots, N-1$, if $0 < x_n < \beta x_k$, then

$$J_n^{x_k}(x_n) = p^+ J_{n+1}^{x_k}(x_n+1) + (1-p^+-p^-) J_{n+1}^{x_k}(x_n) + p^- J_{n+1}^{x_k}(x_n-1), \tag{1.75}$$

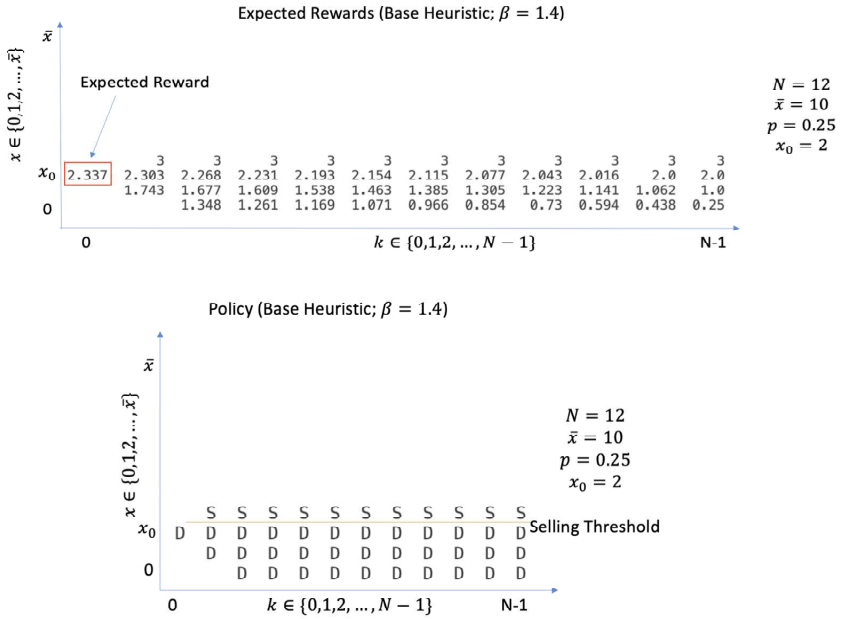

Figure 1.8.3 Table of rewards-to-go for the base policy with $\beta = 1.4$, starting from x_0 [cf. the algorithm (1.74)-(1.77) for $k = 0$].

if $x_n = 0$, then

$$J_n^{x_k}(0) = p^+ J_{n+1}^{x_k}(1) + (1 - p^+) J_{n+1}^{x_k}(0), \tag{1.76}$$

and if $x_n \geq \beta x_k$, then

$$J_n^{x_k}(x_n) = x_n. \tag{1.77}$$

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 policy.

While the reward-to-go for the base policy 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 policy'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 base policy determines the rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, 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

$$p^+ J_{k+1}^{x_k+1}(x_k+1) + (1-p^+-p^-)J_{k+1}^{x_k}(x_k) + p^- J_{k+1}^{x_k-1}(x_k-1) < x_k,$$

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 policy $J_k^{x_k}(x_k)$, 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 policy. 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 policy. In particular, starting from x_0, the rollout policy attains reward 2.388, and the base policy attains reward 2.337. The optimal policy attains reward 2.4. The rollout policy reward is much closer to the optimal than the base policy 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 policy 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 2.291 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 2.375 [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 policy starting from x_0. With the more accurate 200-sample simulation, the rollout algorithm performs better than the base policy starting from x_0, and performs

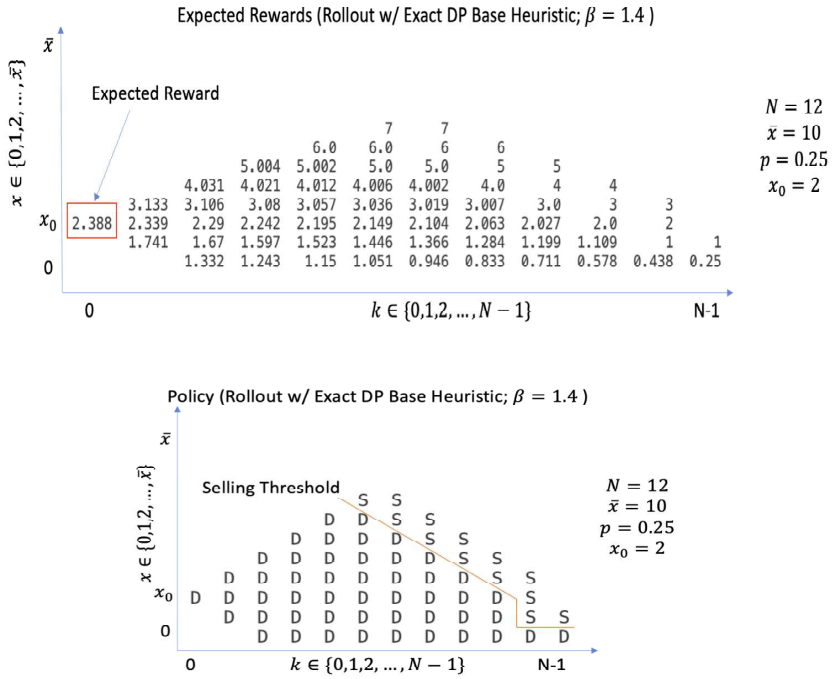

Figure 1.8.4 Table of values of reward-to-go and decisions applied by the rollout policy that corresponds to the base policy with $\beta = 1.4$.

nearly as well as the optimal policy (but still somewhat worse than in the case where exact values of the needed base policy rewards are used (based on an "infinite" number Monte Carlo samples).

It is worth noting here that the heuristic, despite the name "base policy" that we have used, 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.

(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 policy 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 $2\ell+1$.

1.3 (Computational Exercise - Spiders and Flies)

Consider the spiders and flies problem of Example 1.6.4 with two differences: the five flies are stationary (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. 1.6.7. 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.

1.4 (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 as in part (a), and the performance of the algorithm that uses on-line replanning, whereby the optimal policy π^* is recalculated each time the velocity profile changes. Compare with the performance of the rollout policy $\tilde{\pi}$ that uses π as the base policy and on-line replanning.

1.5 (Computational Exercise - Parking Problem)

In reference to Example 1.6.3, a driver aims to park at an inexpensive space on the way to his destination. There are L parking spaces available and a garage at the end. The driver can move in either direction. For example if he is in space i he can either move to $i-1$ with a cost $t-i$, or to $i+1$ with a cost $t+i$, or he can park at a cost $c(i)$ (if the parking space i is free). The only exception is when he arrives at the garage (indicated by index N) and he has to park there at a cost C. Moreover, after the driver visits a parking space he remembers its free/taken status and has an option to return to any parking space he has already visited. However, the driver must park within a given number of stages N, so that the problem has a finite horizon. The initial probability of 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.

1.6 (Computational Exercise - PI and Newton's Method for Linear Quadratic Problems)

The purpose of this exercise is to demonstrate the fast convergence of the PI algorithm. Consider the undiscounted deterministic one-dimensional linear quadratic problem for $a = 1$ and $q = 1$.

(a) Verify that the Bellman equation,

$$Kx^2 = \min_u \left[x^2 + ru^2 + K(x + bu)^2\right],$$

can be written as the equation $F(K) = 0$, where

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

(b) Verify computationally that

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

for the two cases where $r = 0.5$ and b varies, and $b = 2$ and r varies. Here for a given initial state, $\tilde{J}$ is the rollout performance, J^* is the optimal performance, and J is the performance of the base policy $\mu_L(x) = Lx$, where L is given by Eq. (1.68).

(c) *Rollout cost improvement over base policy in adaptive control*: Consider a range of values of b, r, and L_0, and study computationally the effect of the second derivative of F on the ratio K_1/K_0 of rollout to base policy costs, and on the ratio K_1/K^* of rollout to optimal policy costs.

1.7 (Post-Decision States)

The purpose of this exercise is to demonstrate a type of DP simplification that arises often (see [Ber12], Section 6.1.5 for further discussion). Consider the finite horizon stochastic DP problem and assume that the system equation has a special structure whereby from state x_k after applying u_k we move to an intermediate "post-decision state"

$$y_k = p_k(x_k, u_k)$$

at cost $g_k(x_k, u_k)$. Then from y_k we move at no cost to the new state x_{k+1} according to

$$x_{k+1} = h_k(y_k, w_k) = h_k(p_k(x_k, u_k), w_k), \tag{1.78}$$

where the distribution of the disturbance w_k depends only on y_k, and not on prior disturbances, states, and controls. Denote by $J_k(x_k)$ the optimal cost-to-go starting at time k from state x_k, and by $V_k(y_k)$ the optimal cost-to-go starting at time k from post-decision state y_k.

(a) Use Eq. (1.78) to verify that a DP algorithm that generates only J_k is given by

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} \Big[g(x_k, u_k) + E_{w_k} \Big\{ J_{k+1}\big(h_k(p_k(x_k, u_k), w_k)\big) \Big\} \Big].$$

(b) Show that a DP algorithm that generates both J_k and V_k is given by

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} \Big[g(x_k, u_k) + V_k\big(p_k(x_k, u_k)\big) \Big],$$

$$V_k(y_k) = E_{w_k} \Big\{ J_{k+1}\big(h_k(y_k, w_k), w_k\big) \Big\}.$$

(c) Show that a DP algorithm that generates only V_k for all k is given by

$$V_k(y_k) = E_{w_k} \Big\{ \min_{u_{k+1} \in U_{k+1}(h_k(y_k, w_k))} \big[g_{k+1}(h_k(y_k, w_k), u_{k+1}) + V_{k+1}(p_{k+1}(h_k(y_k, w_k), u_{k+1})) \big] \Big\}.$$

2

Principles of Approximation in Value Space

Contents

2.1. Approximation in Value and Policy Space . . . . . p. 114
2.1.1. Approximation in Value Space - One-Step and Multistep Lookahead . . . . . p. 115
2.1.2. Approximation in Policy Space . . . . . p. 119
2.1.3. Combined Approximation in Value and Policy Space . . . . . p. 120
2.2. Off-Line Training, On-Line Play, and Newton's Method p. 125
2.2.1. Approximation in Value Space and Newton's Method . . . . . p. 132
2.2.2. Region of Stability . . . . . p. 136
2.2.3. Policy Iteration, Rollout, and Newton's Method . p. 141
2.2.4. How Sensitive is On-Line Play to the Off-Line Training Process? . . . . . p. 147
2.2.5. Why Not Just Train a Policy Network and Use it Without On-Line Play? . . . . . p. 149

As we noted in Chapter 1, the exact solution of optimal control problems by DP is often impossible in practice. To a great extent, the reason lies in what Bellman has called the "curse of dimensionality." This refers to a rapid increase of the required computation and memory storage as the size of the problem increases. Moreover, there are many circumstances where the structure of the given problem is known well in advance, but some of the problem data, such as various system parameters, may be unknown until shortly before control is needed, thus seriously constraining the amount of time available for the DP computation. The same is true when the system parameters change while we are in the process of applying control, in which case on-line replanning is needed. These difficulties motivate suboptimal control schemes that strike a reasonable balance between convenient implementation and adequate performance.

We have already provided in Chapter 1 a summary of some of the main approximation ideas in RL. In this chapter, we provide a more detailed presentation, focusing primarily on finite horizon methods. Much of our methodology can be adapted to infinite horizon DP. Our discussion will center on the key ideas of approximation in value space, one-step and multistep lookahead, and policy improvement by rollout, as well as various possibilities for their implementation, including off-line training.

2.1 APPROXIMATION IN VALUE AND POLICY SPACE

There are two general types of approximation in DP-based suboptimal control. The first is *approximation in value space*, where we aim to approximate the optimal cost function or the cost function of a given policy, often using some process based on data collection. The second is *approximation in policy space*, where we select a policy from a suitable class of policies based on some criterion; the selection process often uses data, optimization, and neural network approximations. In some settings the value space and policy space approximation approaches may be combined.

In this section we provide a broad overview of the main ideas for these two types of approximation. In this connection, it is important to keep in mind the two types of algorithms underlying AlphaZero and related game programs that we discussed in Section 1.1. The first type is *off-line training*, which computes cost functions and policies before the actual control process begins; for example by using data and training algorithms for neural networks. The second type is *on-line play*, which selects and applies controls during the actual control process of the system. Approximation in policy space is strictly an off-line training-type of algorithm. On the other hand, approximation in value space is primarily an on-line play-type of algorithm, which however, may use cost functions and policies obtained by extensive off-line training.

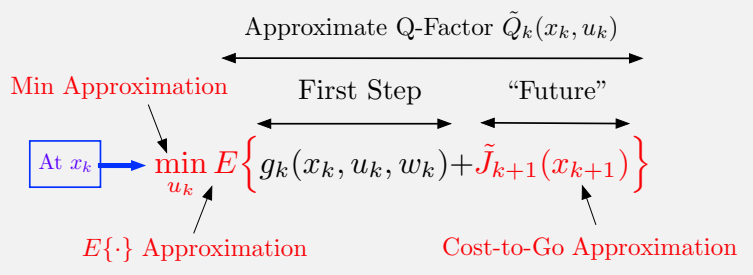

Figure 2.1.1 Schematic illustration of one-step lookahead and the three principal types of approximations. At each state x_k, it uses the control obtained from the minimization

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

or equivalently in terms of Q-factors,

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

This defines the suboptimal policy

$$\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\},$$

referred to as the *one-step lookahead policy based on* $\tilde{J}_{k+1}$, $k = 0, \ldots, N-1$. There are three potential areas of approximation here, which can be considered independently of each other: cost-to-go approximation, expected value approximation, and minimization approximation.

2.1.1 Approximation in Value Space - One-Step and Multistep Lookahead

Let us consider the finite horizon stochastic DP problem of Section 1.2. In the principal form of approximation in value space discussed in these notes, we replace the optimal cost-to-go function J^*_{k+1} in the DP equation with $\tilde{J}_{k+1}$. In particular, at state x_k, we use the control obtained from the minimization

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big\}. \quad (2.1)$$

This process defines a suboptimal policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ and will often be referred to as *one-step lookahead*; see Fig. 2.1.1. There are several possibilities for selecting or computing the functions $\tilde{J}_{k+1}$, many of which are discussed in this class. In some schemes the expected value and minimization operations may also be carried out approximately; see Fig. 2.1.1. For example, we discussed the possibility of approximate minimization for multiagent problems in Section 1.6.

Note that the expected value expression appearing in the right-hand side of Eq. (2.1) can be viewed as an approximate Q-factor,

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

and the minimization in Eq. (2.1) can be written as

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

This suggests a variant of approximation in value space, which is based on using Q-factor approximations that may be obtained directly, i.e., without the intermediate step of obtaining the cost function approximations $\tilde{J}_k$. We will focus primarily on cost function approximation, but we will occasionally digress to discuss direct Q-factor approximation.

Multistep Lookahead

An important extension of one-step lookahead is *multistep lookahead* (also referred to as *ℓ-step lookahead*), whereby at state x_k we minimize the cost of the first $\ell>1$ stages with the future costs approximated by a function $\tilde{J}_{k+\ell}$. For example, in two-step lookahead the function $\tilde{J}_{k+1}$ is given by

$$\tilde{J}_{k+1}(x_{k+1})=\min_{u_{k+1}\in U_{k+1}(x_{k+1})}E\Big\{g_{k+1}(x_{k+1},u_{k+1},w_{k+1})+\tilde{J}_{k+2}\big(f_{k+1}(x_{k+1},u_{k+1},w_{k+1})\big)\Big\},$$

where $\tilde{J}_{k+2}$ is some approximation of the optimal cost-to-go function J^*_{k+2}. More generally, at state x_k we solve the ℓ-stage problem

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}}E\left\{g_k(x_k,u_k,w_k)+\sum_{t=k+1}^{k+\ell-1}g_t\big(x_t,\mu_t(x_t),w_t\big)+\tilde{J}_{k+\ell}(x_{k+\ell})\right\}$$

to obtain a corresponding optimal sequence $\{\tilde{u}_k,\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}\}$. We then use the first control $\tilde{u}_k$ in this sequence, discard $\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}$, obtain the next state

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

and repeat the process with x_k replaced by x_{k+1}; see Fig. 2.1.2.

Actually, one may view ℓ-step lookahead as the special case of one-step lookahead where the lookahead function is the optimal cost function of an $(\ell-1)$-stage DP problem with a terminal cost $\tilde{J}_{k+\ell}(x_{k+\ell})$ on the state $x_{k+\ell}$ obtained after $\ell-1$ stages. However, it is often important to view ℓ-step lookahead separately, in order to address special implementation issues that do not arise in the context of one-step lookahead. Moreover, it

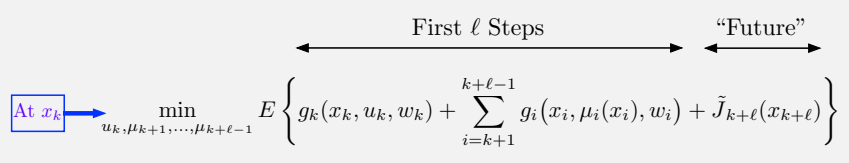

Figure 2.1.2 Schematic illustration of ℓ-step lookahead with approximation in value space. At each state x_k, we solve an ℓ-stage DP problem to obtain a sequence $\{\tilde{u}_k, \tilde{\mu}_{k+1}, \ldots, \tilde{\mu}_{k+\ell-1}\}$, and then use the first control $\tilde{u}_k$ in this sequence. This involves the same three approximations as one-step lookahead: cost-to-go approximation, expected value approximation, and minimization approximation. The minimization of the expected value is more time consuming, but the cost-to-go approximation after ℓ need not be chosen as accurately/carefully as one-step lookahead.

is important to understand how the choice ℓ affects the performance of the ℓ-step lookahead policy.

The motivation for ℓ-step lookahead is that *by increasing the value of ℓ, one may require a less accurate approximation $\tilde{J}_{k+\ell}$* to obtain good performance. Otherwise expressed, for the same quality of cost function approximation, better performance may be obtained as ℓ becomes larger. This makes intuitive sense, since with multistep lookahead, the cost of more stages is treated with optimization exactly, and is also supported by error bounds given in the books [Ber19a], [Ber20a]. Moreover, after many stages, due to randomness, discounting, or other factors, the cost of the remaining stages may become negligible or may not depend much on the choice of the control u_k at the current stage k. Indeed, in practice, longer lookahead results in better performance, although one can construct artificial examples where this is not so (see [Ber19a], Section 2.2.1). In Section 2.2 we will also aim to understand how the length of lookahead plays an important role in the context of Newton step-based visualizations of approximation in value space.

Note that in a deterministic setting, the lookahead problems are also deterministic, and may be addressed by efficient shortest path methods. This makes deterministic problems particularly good candidates for the use of multistep lookahead. Generally, the implementation of ℓ-step lookahead can be prohibitively time-consuming for stochastic problems, because it requires at each step the solution of a stochastic DP problem with an ℓ-step horizon. As a practical guideline, one should at least try to use the largest value of ℓ for which the computational overhead for solving the ℓ-step lookahead minimization problem on-line is acceptable.

In our discussion of approximation in value space of the present section, we will focus primarily on one-step lookahead. Usually, there are straightforward extensions of the main ideas to the multistep context.

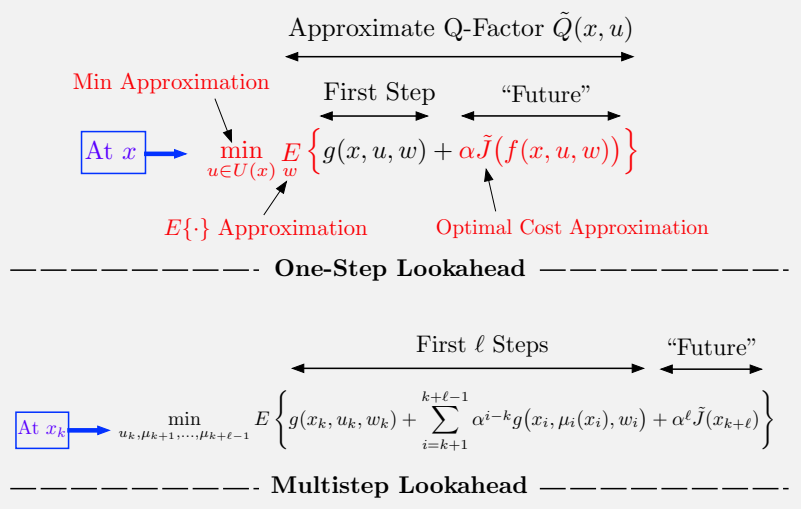

Figure 2.1.3 Schematic illustration of approximation in value space with one-step and multistep lookahead for infinite horizon problems, and the associated three approximations: cost-to-go approximation, expected value approximation, and minimization approximation.

Infinite Horizon Problems

We have provided an introductory discussion of infinite horizon problems, and their basic analytical and algorithmic theory in Section 1.4. In this chapter we will focus on finite horizon problems. However, approximation in value space, with both one-step and multistep lookahead is conceptually very similar in infinite horizon problems. This is convenient, as it will allow us to develop much of the approximation methodology within the conceptually simpler finite horizon context, which is also suited for our discussion of discrete optimization problems.

Regarding approximation in value space, there are three potential areas of approximation for infinite horizon problems, which can be considered independently of each other: cost-to-go approximation, expected value approximation, and minimization approximation; cf. Fig. 2.1.3. This is similar to the finite horizon case; cf. Fig. 2.1.1.

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. Moreover, 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, and their variants are some of these (we will only discuss approximate policy iteration in this book). 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.

2.1.2 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,

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

where r_k is a parameter, and then estimate the parameters r_k using some type of training process or optimization; cf. Fig. 2.1.4.

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 later is primarily an off-line training method (which may be used optionally to provide a policy for on-line rollout).

Neural networks 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. Later in this course 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. An important advantage of approximation in policy space is that once the parametrized policy 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 (2.1). 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.

Later in this course, we will discuss in some detail the approximation of policies for the case where the number of controls available at x_k is finite. In this case, we will see that $\tilde{\mu}_k(x_k, r_k)$ is computed as a randomized policy, i.e., a set of probabilities of applying each of the available controls at x_k (a parametrized policy may be computed in randomized form for reasons of algorithmic/training convenience; in practice it is typically implemented by applying at state x_k the control of maximum probability; for example the training algorithm of AlphaZero uses randomized policies). The methods to obtain the parameter r_k are similar to classification methods used in pattern recognition. This is not surprising because when the control space

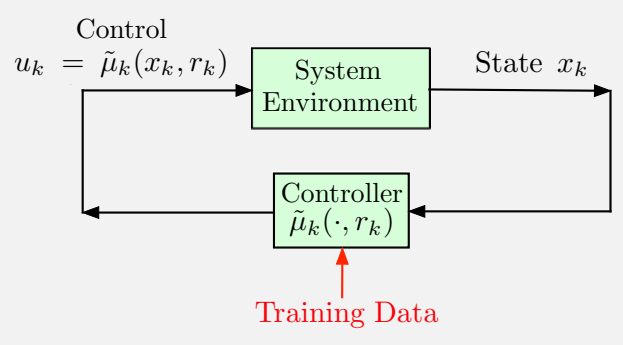

Figure 2.1.4 Schematic illustration of parametric approximation in policy space. A policy

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

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.

is finite, it is possible to view different controls as distinct categories of objects, and to *view any policy* $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$ *as a classifier that assigns a state* x_k *at time* k *to category* $\mu_k(x_k)$.

There are also alternative optimization-based approaches, where 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 not be discussed in this book (for details and many references to the literature, see the RL book [Ber19a], Section 5.7). Actually, this type of approach is used most often in an infinite horizon context, where the policies of interest are stationary, so a single function μ needs to be approximated rather than the N functions $\mu_0, \ldots, \mu_{N-1}$. In this chapter, we focus on finite horizon problems, postponing the discussion of infinite horizon problems for later. Many of the finite horizon methods, however, also apply with small modifications to infinite horizon problems.

2.1.3 Combined Approximation in Value and Policy Space

In this section, we discuss various ways to combine approximation in value and in policy space. In particular, we first describe how approximation in policy space can be built starting from approximation in value space. We then describe a reverse process, namely how we can start from some policy, and construct an approximation in value or Q-factor space, which in turn can be used to construct a new policy through one-step or multistep lookahead. This is the rollout approach, which we will discuss at length in this

chapter and the next one. Finally, we show how to combine the two types of approximation in a perpetual cycle of repeated approximations in value and policy space. This involves the use of approximation architectures, such as neural networks, which we will consider later in this course.

From Values to Policies

A general scheme for parametric approximation in policy space is to obtain 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

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

(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.

A special case of the above procedure, which connects with approximation in value space, is to generate the sample state-control pairs (x_k^s, u_k^s) through a one-step lookahead minimization of the form

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

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

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

† Throughout this book $\|\cdot\|$ 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. 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 will be described later in this course. 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 $\tilde{Q}_k$ is a separately obtained Q-factor approximation. We may view this as *approximation in policy space built on top of approximation in value space*.

From Policies to Values to New Policies - Rollout

An important approach for approximation in value space is to use one-step or ℓ-step lookahead with cost function approximation $\tilde{J}_{k+\ell}(x_{k+\ell})$ equal to the tail problem cost $J_{k+\ell,\pi}(x_{k+\ell})$ starting from $x_{k+\ell}$ and using some known policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$. Thus, in the case of one-step lookahead, we use the control

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1,\pi}\big(f_k(x_k, u_k, w_k)\big) \Big\}.$$

Equivalently, we can use one-step lookahead with the Q-factors $Q_{k,\pi}(x_k, u_k)$ of the policy, i.e., use the control

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U_k(x_k)} Q_{k,\pi}(x_k, u_k).$$

This has the advantage of simplifying the one-step lookahead minimization. In practice, for computational expediency, an approximation to $J_{k,\pi}(x_k)$ or $Q_{k,\pi}(x_k, u_k)$ is often used instead, as we will discuss shortly.

Using the cost function or the Q-factors of a policy as a basis for approximation in value space and obtaining a new policy by lookahead minimization, constitutes the *rollout algorithm*. This is one of the principal subjects of this book. When the values $J_{k+1,\pi}(x_{k+1})$ are not available analytically (which is the typical case), it is necessary to compute them as needed by some form of simulation. In particular, for a deterministic finite horizon problem, we may compute $J_{k+1,\pi}(x_{k+1})$ by accumulating the stage costs along the (unique) trajectory that starts at x_{k+1} and uses π to the end of the horizon. For a stochastic problem it is necessary to obtain $J_{k+1,\pi}(x_{k+1})$ by Monte Carlo simulation, i.e., generate a number of random trajectories starting from x_{k+1} and using π up to the end of the horizon, and then average the corresponding random trajectory costs.

Thus, starting with a policy π, which we will call the *base policy*, the process of one-step or multistep lookahead with cost approximations $J_{k,\pi}$, defines a new policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, which we will call the *rollout policy*; see Fig. 2.1.5. One of the fundamental facts in DP is that the rollout policy has a *policy improvement property*: $\tilde{\pi}$ has no worse cost than π, i.e.,

$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k), \tag{2.2}$$

for all x_k and k. We will discuss this property later in this class.

Since we generally cannot expect to be able to compute the base policy cost function values $J_{k,\pi}(x_k)$ at all states x_k, we may use instead cost

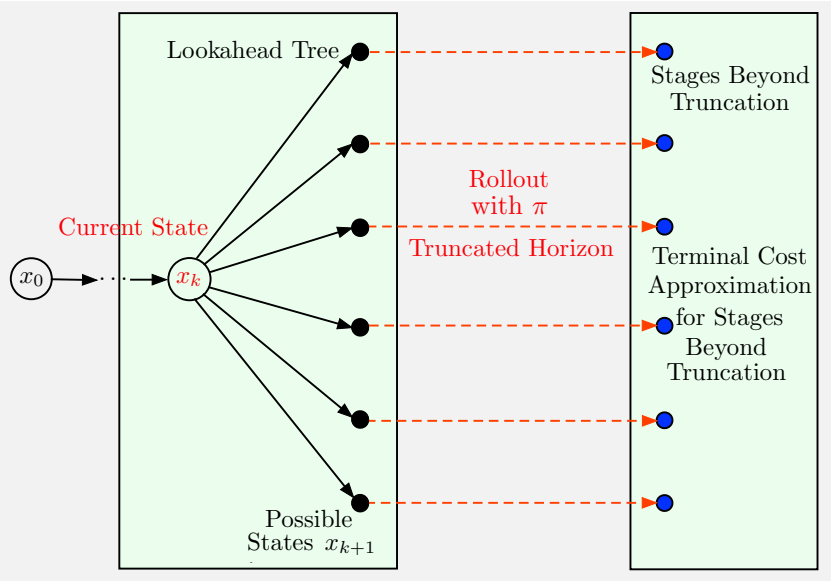

Figure 2.1.5 Schematic illustration of rollout, which involves approximation in value space using a base policy

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

We use one-step or ℓ-step lookahead where $\tilde{J}_{k+\ell}(x_{k+\ell})$ is equal to the tail problem cost $J_{k+\ell,\pi}(x_{k+\ell})$ starting from $x_{k+\ell}$ and using policy π. To economize in computation cost, we may use truncated rollout, whereby we perform the simulation with π for a limited number of stages starting from each possible next state $x_{k+\ell}$, and either neglect the costs of the remaining stages or add some heuristic cost approximation at the end to compensate for these costs. The figure illustrates the case $\ell = 1$.

function approximations that are constructed from data. One possibility is to use Monte Carlo simulation to collect many pairs of state and base policy costs $\big(x_k^s, J_{k,\pi}(x_k^s)\big)$, from which to obtain cost function approximations $\tilde{J}_k$, for each of the stages $k = 1, \ldots, N$, through some form of training process. The functions $\tilde{J}_k$ approximate the base policy cost functions $J_{k,\pi}$, thus yielding an approximate rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ through one-step or multistep lookahead. This policy satisfies the cost improvement property (2.2) in an approximate sense (within some error bound, which is small if $\tilde{J}_k$ is close to $J_{k,\pi}$; see the RL textbook [Ber19a], Section 5.2.6, for more precise statements).

Let us also note the computationally expedient possibility of *truncated rollout*, which is particularly useful for problems with a long horizon; see Fig. 2.1.5. This is to perform the simulation with π for a limited number of stages, and either neglect the costs of the remaining stages or use some

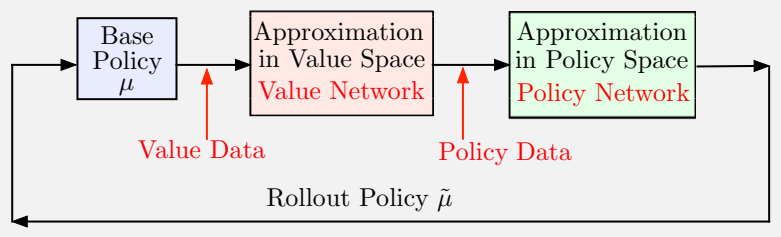

Figure 2.1.6 Schematic illustration of sequential approximation in value and policy space (or perpetual rollout). It produces a sequence of policies and their cost function approximations. Each generated policy is viewed as the rollout policy with the preceding policy viewed as the base policy. The rollout policy is then approximated in policy space, and viewed as the base policy for the next iteration. The approximation in value space may involve either costs or Q-factors of the base policy.

heuristic cost approximation at the end to compensate for these costs. The terminal cost approximation could itself be an approximation of the cost function of the base policy. We will discuss truncated rollout later.

Perpetual Rollout and Approximate Policy Iteration

The rollout process starts with a base policy, which, through approximation in value space, can generate state-control samples of the rollout policy. Once this is done, the rollout policy may be implemented by approximation in policy space and training using state-control pairs generated by the rollout policy, as discussed earlier. Thus the rollout process can be repeated in perpetuity, so we can obtain a sequence of policies (through approximation in policy space) and corresponding sequence of cost approximations (through approximation in value space); see Fig. 2.1.6.

When neural networks are used, the approximations in value and policy space are commonly referred to as the *value network* and the *policy network*, respectively. For example, the AlphaGo and AlphaZero programs ([SHM16], [SHS17], [SHS17]), use both value and policy networks for approximation in value space and policy space. Note that the value and policy networks must be constructed off-line (before the control process begins), since their training involves a lot of data collection and computation.

The process just described also applies and indeed becomes simpler for infinite horizon problems. It underlies the class of *approximate policy iteration* methods, which we discussed briefly in Chapter 1, and we will revisit later in this course. An important type of such a method is known as *optimistic approximate policy iteration*, where the approximations in value and policy space are done using limited amounts of data (e.g., use just a few samples of state-control pairs to perform one or more gradient-type iterations to update the parameters of a value network and/or a policy network). Methods of this type include algorithms such as *Q-learning* and

policy evaluation by *temporal differences*. Together with their variations, which depend on the details of the data collection, and the amount of data used for the value and policy space approximations, these algorithms underlie a large part of the RL methodology.

2.2 OFF-LINE TRAINING, ON-LINE PLAY, AND NEWTON'S METHOD

In this section we will focus on infinite horizon problems, as introduced in Section 1.4, and we will use geometric constructions to obtain insight into Bellman's equation, the value and policy iteration algorithms, approximation in value space, and some of the properties of the corresponding one-step or multistep lookahead policy $\tilde{\mu}$. To understand these constructions, we focus on infinite horizon problems, and we will use the abstract notational framework that we introduced briefly in Chapter 1. In particular, we denote by TJ the function of x that appears in the right-hand side of Bellman's equation. Its value at state x is given by†

$$(TJ)(x) = \min_{u \in U(x)} E\Big\{ g(x,u,w) + \alpha J\big(f(x,u,w)\big) \Big\}, \qquad \text{for all } x. \quad (2.3)$$

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

$$(T_\mu J)(x) = E\Big\{ g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big) \Big\}, \qquad \text{for all } x. \quad (2.4)$$

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

In Chapter 1 we elaborated on visualizations of a special case of the Bellman operators, namely the Riccati operators that arise in the context of the linear quadratic problem. As discussed in Section 1.5, the Riccati

† Recall here our convention that we will be using "min" instead of the more formal "inf," even we are not sure that minimum is attained.

‡ Within the context of this work, the functions J on which T and T_μ operate will be real-valued functions of x. We will assume throughout that the expected values in Eqs. (2.3) and (2.4) are well-defined and finite when J is real-valued. This implies that $T_\mu J$ will also be real-valued functions of x. On the other hand $(TJ)(x)$ may take the value $-\infty$ because of the minimization in Eq. (2.3). We allow this possibility, although our illustrations will primarily depict the case where TJ is real-valued. Note that the general theory of abstract DP is developed with the use of extended real-valued functions; see the abstract DP book [Ber22a].

operators are simply the Bellman operators, restricted to quadratic functions J. In this section we extend these visualization ideas to the general infinite horizon problem of Section 1.4. Despite the restrictive framework of our analysis of the linear quadratic problem of Section 1.5, we will argue that the principal algorithmic insights relating to approximation in value space, rollout, policy iteration, and Newton's method survive intact within the general infinite horizon context.

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

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

then we have

$$(TJ)(x) \geq (TJ')(x), \qquad (T_\mu J)(x) \geq (T_\mu J')(x), \qquad \text{for all } x \text{ and } \mu. \tag{2.5}$$

This is evident from Eqs. (2.3) and (2.4).

Another important property is that the Bellman operator T_μ is *linear*, in the sense that it has the form $T_\mu J = G + A_\mu J$, where $G \in R(X)$ is some function and $A_\mu : R(X) \mapsto R(X)$ is an operator such that for any functions J_1, J_2, and scalars γ_1, γ_2, we have†

$$A_\mu(\gamma_1 J_1 + \gamma_2 J_2) = \gamma_1 A_\mu J_1 + \gamma_2 A_\mu J_2.$$

Moreover, from the definitions (2.3) and (2.4), we have

$$(TJ)(x) = \min_{\mu \in \mathcal{M}} (T_\mu J)(x), \qquad \text{for all } x,$$

where $\mathcal{M}$ is the set of stationary policies. This is true because for any policy μ, there is no coupling constraint between the controls $\mu(x)$ and $\mu(x')$ that correspond to two different states x and x'. It follows that $(TJ)(x)$ *is a concave function of* J *for every* x, something that will be important for our interpretation of one-step and multistep lookahead as a Newton iteration for solving the Bellman equation $J = TJ$.

Example 2.2.1 (A Two-State and Two-Control Example)

Assume that there are two states 1 and 2, and two controls u and v. Consider the policy μ that applies control u at state 1 and control v at state 2. Then the operator T_μ takes the form

$$(T_\mu J)(1) = \sum_{y=1}^{2} p_{1y}(u)\big(g(1,u,y) + \alpha J(y)\big), \tag{2.6}$$

† An operator T_μ with this property is often called "affine," but in this work we just call it "linear." Also we use abbreviated notation to express pointwise equalities and inequalities, so that we write $J = J'$ or $J \geq J'$ to express the fact that $J(x) = J'(x)$ or $J(x) \geq J'(x)$, for all x, respectively.

$$(T_\mu J)(2) = \sum_{y=1}^{2} p_{2y}(v)\big(g(2,v,y) + \alpha J(y)\big), \tag{2.7}$$

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

$$(TJ)(1) = \min\left[\sum_{y=1}^{2} p_{1y}(u)\big(g(1,u,y) + \alpha J(y)\big), \sum_{y=1}^{2} p_{1y}(v)\big(g(1,v,y) + \alpha J(y)\big)\right], \tag{2.8}$$

$$(TJ)(2) = \min\left[\sum_{y=1}^{2} p_{2y}(u)\big(g(2,u,y) + \alpha J(y)\big), \sum_{y=1}^{2} p_{2y}(v)\big(g(2,v,y) + \alpha J(y)\big)\right]. \tag{2.9}$$

Thus, $(TJ)(1)$ and $(TJ)(2)$ are concave and piecewise linear as functions of the two-dimensional vector J (with two pieces; more generally, as many linear pieces as the number of controls). This concavity property holds in general since $(TJ)(x)$ is the minimum of a collection of linear functions of J, one for each $u \in U(x)$. Figure 2.2.1 illustrates $(T_\mu J)(1)$ for the cases where $\mu(1) = u$ and $\mu(1) = v$, $(T_\mu J)(2)$ for the cases where $\mu(2) = u$ and $\mu(2) = v$, $(TJ)(1)$, and $(TJ)(2)$, as functions of $J = \big(J(1), J(2)\big)$.

Mathematically the concavity property of T manifests itself in that the set

$$C = \big\{(J,\xi) \in R(X) \times R(X) \mid (TJ)(x) \geq \xi(x), \text{ for all } x \in X\big\} \tag{2.10}$$

is convex as a subset of $R(X) \times R(X)$, where $R(X)$ is the set of real-valued functions over the state space X. This convexity property is verified by showing that given (J_1, ξ_1) and (J_2, ξ_2) in C, and $\gamma \in [0,1]$, we have

$$\big(\gamma J_1 + (1-\gamma)J_2,\ \gamma\xi_1 + (1-\gamma)\xi_2\big) \in C.$$

The proof of this is straightforward by using the concavity of $(TJ)(x)$ for each x.

Critical properties from the DP point of view are whether T and T_μ have fixed points; equivalently, whether the Bellman equations $J = TJ$ and $J = T_\mu J$ have solutions within the class of real-valued functions, and whether the set of solutions includes J^* and J_μ, respectively. It may thus be important to verify that T or T_μ are contraction mappings. This is true

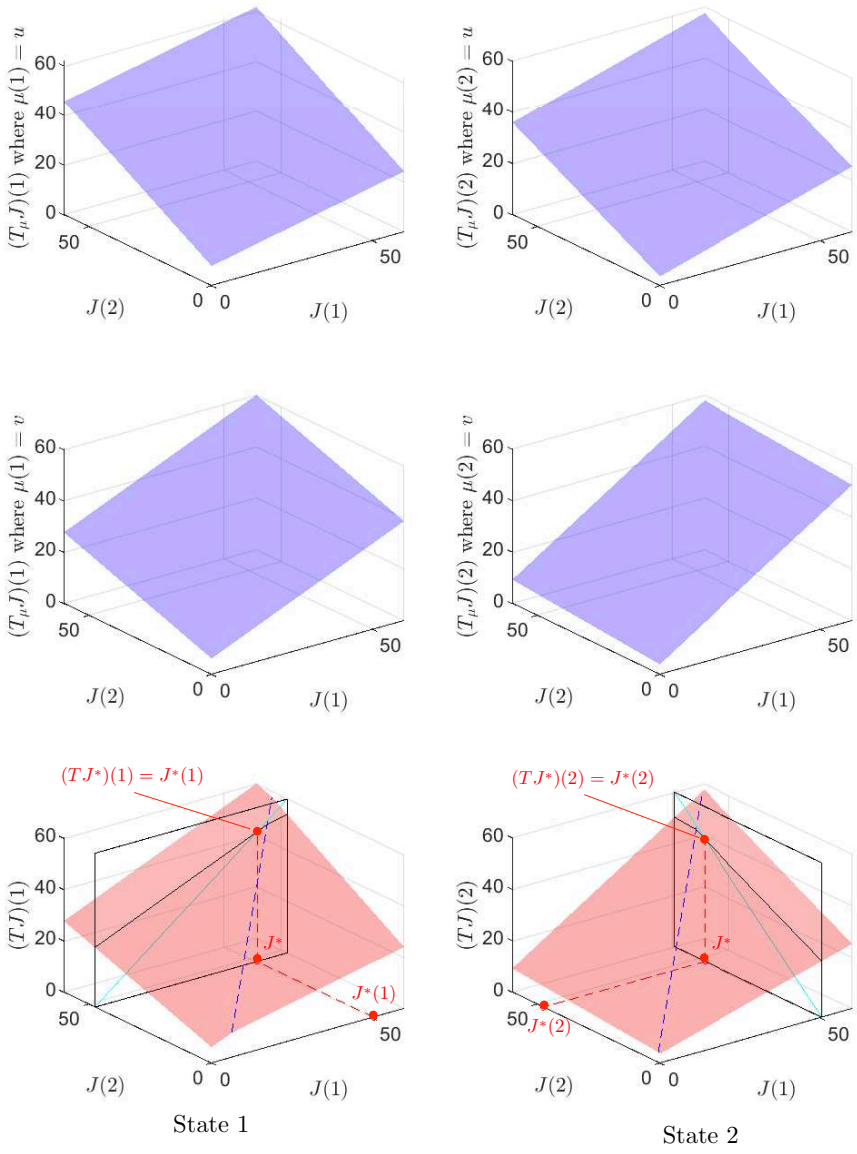

Figure 2.2.1 Geometric illustrations of the Bellman operators T_μ and T for states 1 and 2 in Example 2.2.1; cf. Eqs. (2.6)-(2.9). The problem's transition probabilities are: $p_{11}(u) = 0.3$, $p_{12}(u) = 0.7$, $p_{21}(u) = 0.4$, $p_{22}(u) = 0.6$, $p_{11}(v) = 0.6$, $p_{12}(v) = 0.4$, $p_{21}(v) = 0.9$, $p_{22}(v) = 0.1$. The stage costs are $g(1, u, 1) = 3$, $g(1, u, 2) = 10$, $g(2, u, 1) = 0$, $g(2, u, 2) = 6$, $g(1, v, 1) = 7$, $g(1, v, 2) = 5$, $g(2, v, 1) = 3$, $g(2, v, 2) = 12$. The discount factor is $\alpha = 0.9$, and the optimal costs are $J^*(1) = 50.59$ and $J^*(2) = 47.41$. The optimal policy is $\mu^*(1) = v$ and $\mu^*(2) = u$. The figure also shows two one-dimensional slices of T that are parallel to the $J(1)$ and $J(2)$ axes and pass through J^*.

for example in the benign case of discounted problems with bounded cost per stage. However, for undiscounted problems, asserting the contraction property of T or T_μ may be more complicated, and even impossible; the abstract DP book [Ber18a] deals extensively with such questions, and related issues regarding the solution sets of the Bellman equations.

Geometrical Interpretations

We will now interpret the Bellman operators geometrically, starting with T_μ. Figure 2.2.2 illustrates its form. Note here that the functions J and $T_\mu J$ are multidimensional. They have as many scalar components $J(x)$ and $(T_\mu J)(x)$, respectively, as there are states x, but they can only be shown projected onto one dimension. The function $T_\mu J$ for each policy μ is linear. The cost function J_μ satisfies $J_\mu = T_\mu J_\mu$, so it is obtained from the intersection of the graph of $T_\mu J$ and the 45 degree line, when J_μ is real-valued. Later we will interpret the situation where J_μ is not real-valued with lack of system stability under μ [we have $J_\mu(x) = \infty$ for some initial states x].

The form of the Bellman operator T is illustrated in Fig. 2.2.3. Again the functions J, J^*, TJ, $T_\mu J$, etc, are multidimensional, but they are shown projected onto one dimension (alternatively they are illustrated for a system with a single state, plus possibly a termination state). The Bellman equation $J = TJ$ may have one or many real-valued solutions. It may also have no real-valued solution in exceptional situations, as we will discuss later in this class. The figure assumes a unique real-valued solution of the Bellman equations $J = TJ$ and $J = T_\mu J$, which is true if T and T_μ are contraction mappings, as is the case for discounted problems with bounded cost per stage. Otherwise, these equations may have no solution or multiple solutions within the class of real-valued functions. The equation $J = TJ$ typically has J^* as a solution, but may have more than one solution in cases where either $\alpha = 1$, or $\alpha < 1$ and the cost per stage is unbounded.

Note that the visualizations of the Bellman operators are consistent with the ones for Riccati operators that we gave in Section 1.5. This is also true for subsequent visualizations in this section, involving for example Newton step interpretations for various forms of approximation in value space.

Visualization of Value Iteration

The operator notation simplifies algorithmic descriptions, derivations, and proofs related to DP. For example, we can write the VI algorithm in the compact form

$$J_{k+1} = TJ_k, \qquad k = 0, 1, \ldots,$$

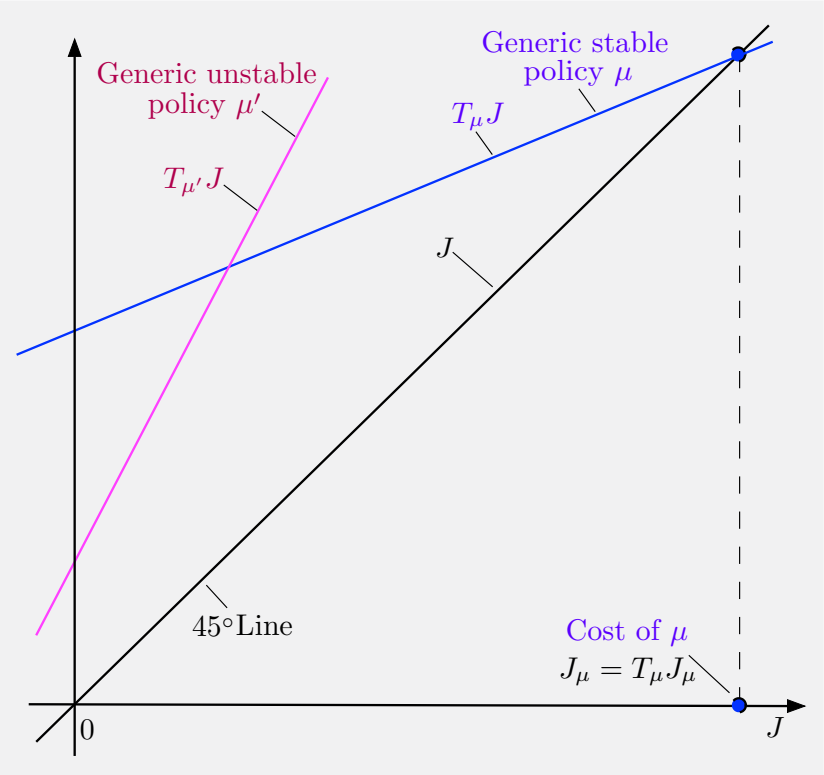

Figure 2.2.2 Geometric interpretation of the linear Bellman operator T_μ and the corresponding Bellman equation. The graph of T_μ is a plane in the space $R(X) \times R(X)$, and when projected on a one-dimensional plane that corresponds to a single state and passes through J_μ, it becomes a line. Then there are three cases:

(a) The line has slope less than 45 degrees, so it intersects the 45-degree line at a unique point, which is equal to J_μ, the solution of the Bellman equation $J = T_\mu J$. This is true if T_μ is a contraction mapping, as is the case for discounted problems with bounded cost per stage.

(b) The line has slope greater than 45 degrees. Then it intersects the 45-degree line at a unique point, which is a solution of the Bellman equation $J = T_\mu J$, but is not equal to J_μ. Then J_μ is not real-valued; we will call such μ *unstable* under μ.

(c) The line has slope exactly equal to 45 degrees. This is an exceptional case where the Bellman equation $J = T_\mu J$ has an infinite number of real-valued solutions or no real-valued solution at all; we will provide examples where this occurs later.

as illustrated in Fig. 2.2.4. Moreover, the VI algorithm for a given policy

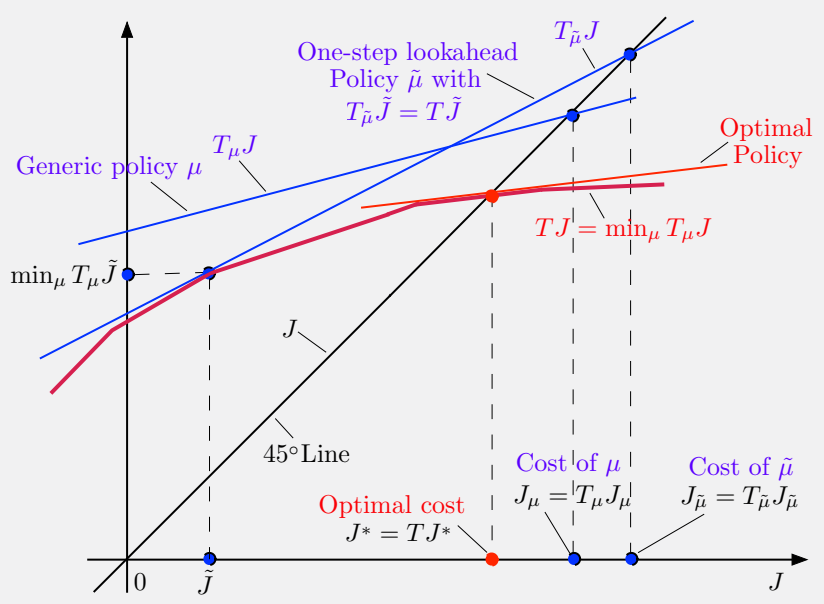

Figure 2.2.3 Geometric interpretation of the Bellman operator T, and the corresponding Bellman equation. For a fixed x, the function $(TJ)(x)$ can be written as $\min_\mu (T_\mu J)(x)$, so it is concave as a function of J. The optimal cost function J^* satisfies $J^* = TJ^*$, so it is obtained from the intersection of the graph of TJ and the 45 degree line shown, assuming J^* is real-valued.

Note that the graph of T lies below the graph of every operator T_μ, and is in fact obtained as the lower envelope of the graphs of T_μ as μ ranges over the set of policies $\mathcal{M}$. In particular, for any given function $\tilde{J}$, for every x, the value $(T\tilde{J})(x)$ is obtained by finding a support hyperplane/subgradient of the graph of the concave function $(TJ)(x)$ at $J = \tilde{J}$, as shown in the figure. This support hyperplane is defined by the control $\mu(x)$ of a policy $\tilde{\mu}$ that attains the minimum of $(T_\mu \tilde{J})(x)$ over μ:

$$\tilde{\mu}(x) \in \arg\min_{\mu \in \mathcal{M}} (T_\mu \tilde{J})(x)$$

(there may be multiple policies attaining this minimum, defining multiple support hyperplanes). This construction also shows how the minimization

$$(T\tilde{J})(x) = \min_{\mu \in \mathcal{M}} (T_\mu \tilde{J})(x)$$

corresponds to a linearization of the mapping T at the point $\tilde{J}$.

μ can be written as

$$J_{k+1} = T_\mu J_k, \qquad k = 0, 1, \ldots,$$

and it can be similarly interpreted, except that the graph of the function $T_\mu J$ is linear. Also we will see shortly that there is a similarly compact description for the policy iteration algorithm.

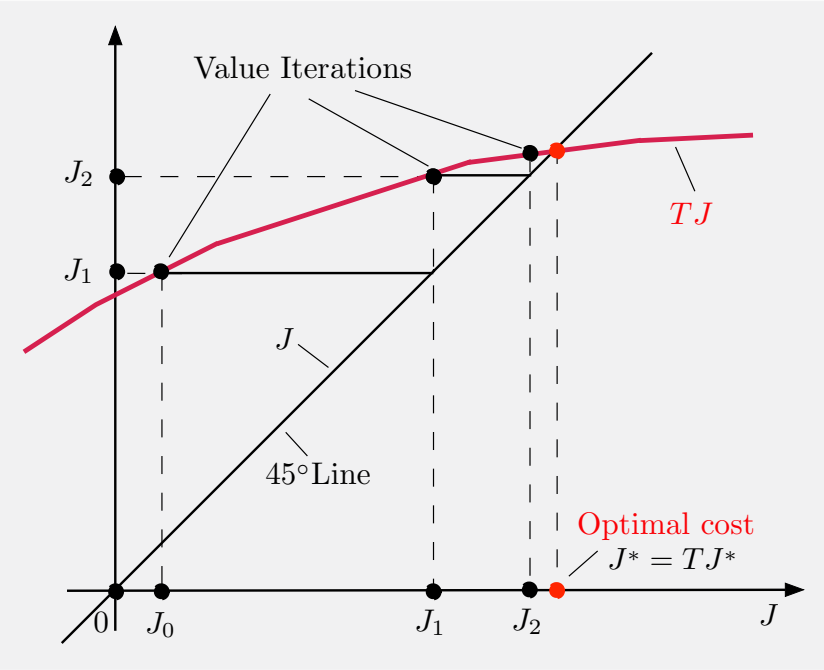

Figure 2.2.4 Geometric interpretation of the VI algorithm $J_{k+1} = TJ_k$, starting from some initial function J_0. Successive iterates are obtained through the staircase construction shown in the figure. The VI algorithm $J_{k+1} = T_\mu J_k$ for a given policy μ can be similarly interpreted, except that the graph of the function $T_\mu J$ is linear.

To keep the presentation simple, we will focus our attention on the abstract DP framework as it applies to the infinite horizon problems introduced in Section 1.4. In particular, we will assume without further mention that T *and* T_μ *have the monotonicity property (2.5), that* $T_\mu J$ *is linear for all* μ*, and that (as a consequence) the component* $(TJ)(x)$ *is concave as a function of* J *for every state* x. We note, however, that the abstract notation facilitates the extension of the infinite horizon DP theory to models beyond the ones that we discuss in this work. Such models include semi-Markov problems, minimax control problems, risk sensitive problems, Markov games, and others (see the DP textbook [Ber12], and the abstract DP monograph [Ber18a]).

2.2.1 Approximation in Value Space and Newton's Method

Let us now consider approximation in value space and an abstract geometric interpretation, first provided in the author's book [Ber20a]. By using the operators T and T_μ, for a given $\tilde{J}$, a one-step lookahead policy $\tilde{\mu}$ is

characterized by the equation

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

as in Fig. 2.2.5. Furthermore, this equation implies that the graph of $T_{\tilde{\mu}}J$ just touches the graph of TJ at $\tilde{J}$, as shown in the figure. In mathematical terms, the set

$$C_{\tilde{\mu}} = \big\{(J,\xi) \mid T_{\tilde{\mu}}J \geq \xi\big\},$$

contains the convex set C of Eq. (2.10) (since $TJ \geq \xi$ implies that $T_{\tilde{\mu}}J \geq \xi$), and has a common point $(\tilde{J}, T_{\tilde{\mu}}\tilde{J})$ with C. Moreover, for each state $x \in X$ the hyperplane $H_{\tilde{\mu}}(x)$

$$H_{\tilde{\mu}}(x) = \Big\{\big(J(x),\xi(x)\big) \mid (T_{\tilde{\mu}}J)(x) = \xi(x)\Big\},$$

supports from above the convex set

$$\Big\{\big(J(x),\xi(x)\big) \mid (TJ)(x) \geq \xi(x)\Big\}$$

at the point $\big(\tilde{J}(x), (T\tilde{J})(x)\big)$ and defines a subgradient of $(TJ)(x)$ at $\tilde{J}$. Note that the one-step lookahead policy $\tilde{\mu}$ need not be unique, since T need not be differentiable.

Thus, the equation $J = T_{\tilde{\mu}}J$ is a pointwise (for each x) linearization of the equation $J = TJ$ at $\tilde{J}$, and its solution, $J_{\tilde{\mu}}$, can be viewed as the result of a Newton iteration at the point $\tilde{J}$. In summary, *the Newton iterate at $\tilde{J}$ is $J_{\tilde{\mu}}$*, the solution of the linearized equation $J = T_{\tilde{\mu}}J$.†

† The classical 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. 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 system at the current iterate, but relax the differentiability requirement to piecewise differentiability, and/or component concavity, while maintaining the superlinear convergence property of the method; see the monograph [Ber22b], and the paper [Ber22c].

As noted earlier, approximation in value space with ℓ-step lookahead using $\tilde{J}$ is the same as approximation in value space with one-step lookahead using the $(\ell-1)$-fold operation of T on $\tilde{J}$, $T^{\ell-1}\tilde{J}$. Thus it can be interpreted as a Newton step starting from $T^{\ell-1}\tilde{J}$, the result of $\ell-1$ value iterations applied to $\tilde{J}$. This is illustrated in Fig. 2.2.6. In this connection, we note that several variants of Newton's method that involve combinations of first-order iterative methods, such as the Gauss-Seidel and Jacobi algorithms, and Newton's method, are well-known in numerical analysis. They belong to the general family of *Newton-SOR methods* (SOR stands for "successive over-relaxation"); see the classic book by Ortega and Rheinboldt [OrR70] (Section 13.4). Their convergence rate is superlinear, similar to Newton's method, as long as they involve a pure Newton step, along with the first-order steps.

Local and Global Performance Estimates Compared

The preceding Newton step interpretation of the move from $\tilde{J}$ (the terminal cost function approximation) to $J_{\tilde{\mu}}$ (the cost function of the lookahead policy $\tilde{\mu}$) suggests a superlinear performance estimate

$$\max_x \big|J_{\tilde{\mu}}(x) - J^*(x)\big| = o\Big(\max_x \big|\tilde{J}(x) - J^*(x)\big|\Big).$$

However, this estimate is *local* in character. It is meaningful only when $\tilde{J}$ is "close" to J^*. When $\tilde{J}$ is far from J^*, the difference $\max_x \big|J_{\tilde{\mu}}(x) - J^*(x)\big|$ may be large and even infinite when $\tilde{\mu}$ is unstable (see the discussion in the next section).

There are global estimates for the difference

$$\max_x \big|J_{\tilde{\mu}}(x) - J^*(x)\big|$$

for several types of problems, including the upper bound

$$\max_x \big|J_{\tilde{\mu}}(x) - J^*(x)\big| \leq \frac{2\alpha^\ell}{1-\alpha} \max_x \big|\tilde{J}(x) - J^*(x)\big|$$

The structure of the Bellman operators (2.3) and (2.4), with their monotonicity and concavity properties, tends to enhance the convergence and rate of convergence properties of Newton's method, even in the absence of differentiability, as evidenced by the convergence analysis of PI, and the extensive favorable experience with rollout, PI, and MPC. In this connection, it is worth noting that in the case of Markov games, where the concavity property does not hold, the PI method may oscillate, as shown by Pollatschek and Avi-Itzhak [PoA69], and needs to be modified to restore its global convergence; see the author's paper [Ber21c].

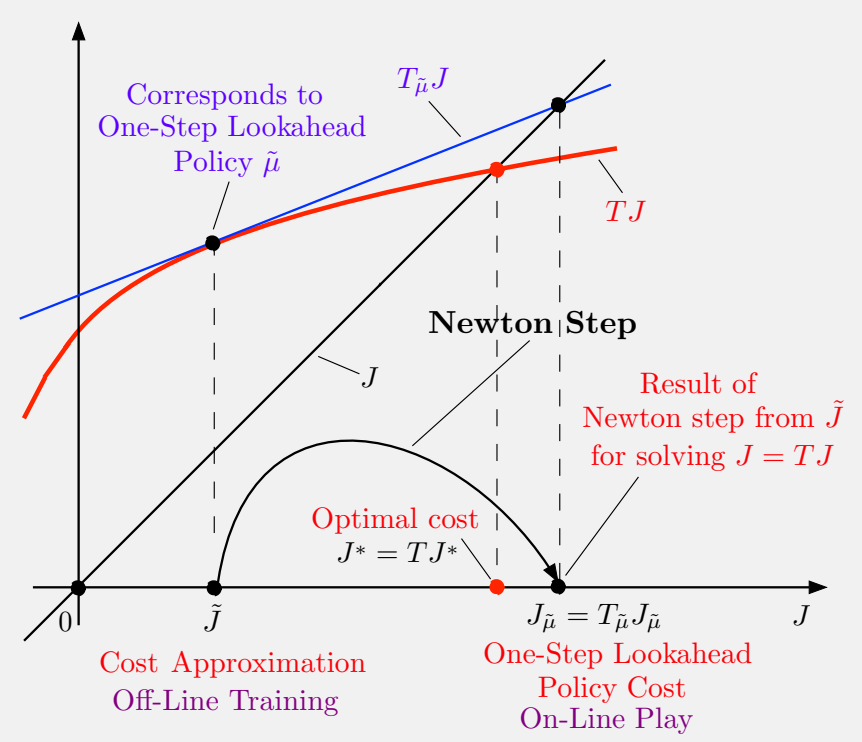

Figure 2.2.5 Geometric interpretation of approximation in value space and the one-step lookahead policy $\tilde{\mu}$ as a step of Newton's method. Given $\tilde{J}$, we find a policy $\tilde{\mu}$ that attains the minimum in the relation $T\tilde{J} = \min_\mu T_\mu \tilde{J}$. This policy satisfies $T\tilde{J} = T_{\tilde{\mu}}\tilde{J}$, so the graph of TJ and $T_{\tilde{\mu}}J$ touch at $\tilde{J}$, as shown. It may not be unique. Because TJ has concave components, the equation $J = T_{\tilde{\mu}}J$ is the linearization of the equation $J = TJ$ at $\tilde{J}$. The linearized equation is solved at the typical step of Newton's method to provide the next iterate, which is just $J_{\tilde{\mu}}$.

for ℓ-step lookahead, and α-discounted problems where all the Bellman operators T_μ are contraction mappings; see the neurodynamic programming book [BeT96] (Section 6.1, Prop. 6.1), or the RL book [Ber20a] (Section 5.4, Prop. 5.4.1). These books also contain other related global estimates, which hold for all $\tilde{J}$, both close and far from J^*. However, these global estimates tend to be overly conservative and not representative of the performance of approximation in value space schemes when $\tilde{J}$ is near J^*. They will not be considered in these notes, as they do not contribute to the insights and methodological points of view that we want to focus on. For example, for finite spaces α-discounted MDP, $\tilde{\mu}$ can be shown to be optimal when $\max_x \left|\tilde{J}(x) - J^*(x)\right|$ is sufficiently small.

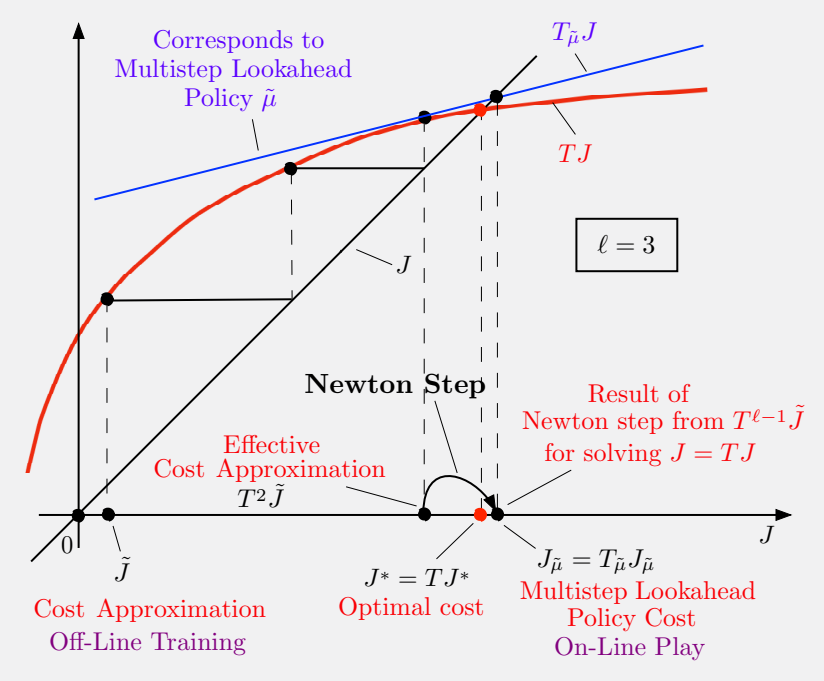

Figure 2.2.6 Geometric interpretation of approximation in value space with ℓ-step lookahead (in this figure $\ell = 3$). It is the same as approximation in value space with one-step lookahead using $T^{\ell-1}\tilde{J}$ as cost approximation. It can be viewed as a Newton step at the point $T^{\ell-1}\tilde{J}$, the result of $\ell - 1$ value iterations applied to $\tilde{J}$. Note that as ℓ increases the cost function $J_{\tilde{\mu}}$ of the ℓ-step lookahead policy $\tilde{\mu}$ approaches more closely the optimal J^*, and that $\lim_{\ell\to\infty} J_{\tilde{\mu}} = J^*$.

2.2.2 Region of Stability

For any control system design method, the stability of the policy obtained is of paramount importance. It is thus essential to investigate and verify the stability of controllers obtained through approximation in value space schemes. Historically, there have been several proposed definitions of stability in control theory. Within the context of this work, our focus on stability issues will be for problems with a termination state t, which is cost-free, and with a cost per stage that is positive outside the termination state, such as the undiscounted positive cost deterministic problem introduced earlier [cf. Eqs. (1.26)-(1.28)]. Moreover, it is best for our purposes to adopt an optimization-based definition. In particular, we say that a policy μ is *unstable* if $J_\mu(x) = \infty$ for some states x. Equivalently, we say that the policy μ *stable* if $J_\mu(x) < \infty$ for all states x. This definition has the advantage that it applies to general state and control spaces. Naturally, it

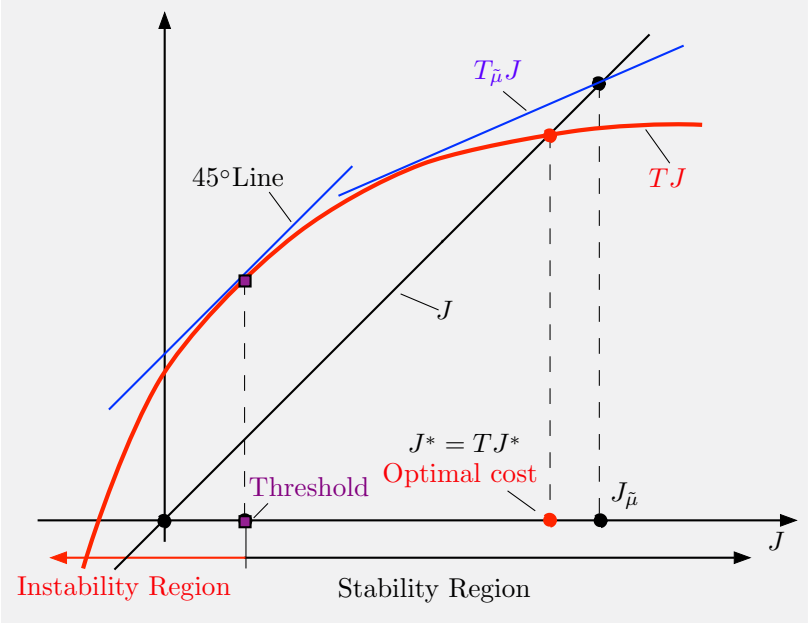

Figure 2.2.7 Illustration of the regions of stability and instability for approximation in value space with one-step lookahead. The stability region is the set of all $\tilde{J}$ such that the policy $\tilde{\mu}$ with $T\tilde{J} = T_{\tilde{\mu}}\tilde{J}$ satisfies $J_{\tilde{\mu}}(x) < \infty$ for all x.

can be made more specific in particular problem instances.†

In the context of approximation in value space we are interested in the *region of stability*, which is the set of cost function approximations $\tilde{J} \in R(X)$ for which the corresponding one-step or multistep lookahead policies $\tilde{\mu}$ are stable. For discounted problems with bounded cost per stage, all policies have real-valued cost functions, so questions of stability do not arise. In general, however, the region of stability may be a strict subset of the set of real-valued functions; this will be illustrated later for the undiscounted deterministic case of the linear quadratic problem of Section 2.1 [cf. Eqs. (1.35), (1.36)]. Figure 2.2.7 illustrates the region of stability for approximation in value space with one-step lookahead.

† For the undiscounted positive cost deterministic problem introduced earlier [cf. Eqs. (1.26)-(1.28)], it can be shown that if a policy μ is stable, then J_μ is the "smallest" solution of the Bellman equation $J = T_\mu J$ within the class of nonnegative real-valued functions, and under mild assumptions it is the unique solution of $J = T_\mu J$ within the class of nonnegative real-valued functions J with $J(t) = 0$; see the author's paper [Ber17b]. Moreover, if μ is unstable, then the Bellman equation $J = T_\mu J$ has no solution within the class of nonnegative real-valued functions.

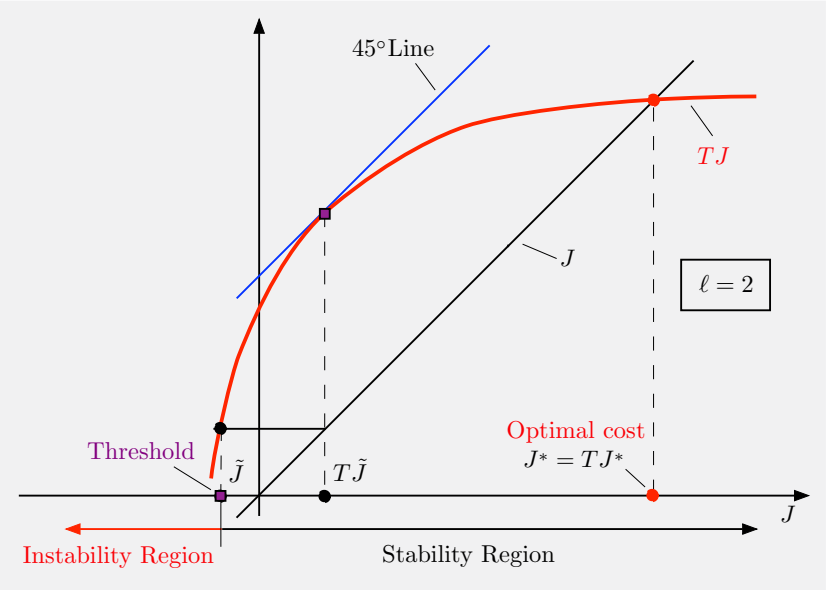

Figure 2.2.8 Illustration of the regions of stability and instability for approximation in value space with multistep lookahead. The stability region is the set of all $\tilde{J}$ for which the policy $\tilde{\mu}$ such that $T^\ell \tilde{J} = T_{\tilde{\mu}} T^{\ell-1} \tilde{J}$ satisfies $J_{\tilde{\mu}}(x) < \infty$ for all x (the figure shows the case $\ell = 2$). The region of instability tends to be reduced as ℓ increases.

An interesting observation from Fig. 2.2.7 is that if $\tilde{J}$ does not belong to the region of stability and $\tilde{\mu}$ is a corresponding one-step lookahead unstable policy, the Bellman equation $J = T_{\tilde{\mu}} J$ may have real-valued solutions. However, these solutions will not be equal to $J_{\tilde{\mu}}$, as this would violate the definition of region of stability. Generally, if T_μ is not a contraction mapping, T_μ may have real-valued fixed points, none of which is equal to J_μ.

Figure 2.2.8 illustrates the region of stability for the case of multistep lookahead. The insights from this figure are similar to the one-step lookahead case of Fig. 2.2.7. However, the figure indicates that *the region of stability of the ℓ-step lookahead controller $\tilde{\mu}$ depends on ℓ, and tends to become larger as ℓ increases*. The reason is that ℓ-step lookahead with terminal cost $\tilde{J}$ is equivalent to one-step lookahead with terminal cost $T^{\ell-1}\tilde{J}$, which tends to be closer to the optimal cost function J^* than $\tilde{J}$ (assuming convergence of the VI method).

How Can we Obtain Function Approximations $\tilde{J}$ Within the Region of Stability?

Naturally, identifying and obtaining cost function approximations $\tilde{J}$ that

lie within the region of stability with either one-step or multistep lookahead is very important within our context. We will focus on this question for the special case where the expected cost per stage is nonnegative

$$E\big\{g(x,u,w)\big\} \geq 0, \qquad \text{for all } x,\ u \in U(x),$$

and assume that J^* is real-valued. This is the case of most interest in model predictive control, but also arises in other problems of interest, including stochastic shortest path problems that involve a termination state.

From Fig. 2.2.8 it can be conjectured that if the sequence $\{T^k\tilde{J}\}$ generated by the VI algorithm converges to J^* for all $\tilde{J}$ such that $0 \leq \tilde{J} \leq J^*$ (which is true under very general conditions; see [Ber12], [Ber18a]), then $T^{\ell-1}\tilde{J}$ belongs to the region of stability for sufficiently large ℓ. Related ideas have been discussed in the adaptive DP literature by Liu and his collaborators [HWL21], [LXZ21], [WLL16], and by Heydari [Hey17], [Hey18], who provide extensive references; see also Winnicki et al. [WLL21]. We will revisit this issue in the context of linear quadratic problems. This conjecture is generally true, but requires that, in addition to J^*, all functions $\tilde{J}$ within a neighborhood of J^* belong to the region of stability. Our subsequent discussion will aim to address this difficulty.

An important fact in our context is that *the region of stability includes all real-valued nonnegative functions $\tilde{J}$ such that*

$$T\tilde{J} \leq \tilde{J}. \tag{2.11}$$

Indeed if $\tilde{\mu}$ is the corresponding one-step lookahead policy, we have

$$T_{\tilde{\mu}}\tilde{J} = T\tilde{J} \leq \tilde{J},$$

and from a well-known result on nonnegative cost infinite horizon problems [see [Ber12], Prop. 4.1.4(a)], it follows that

$$J_{\tilde{\mu}} \leq \tilde{J};$$

(the proof argument is that if $T_{\tilde{\mu}}\tilde{J} \leq \tilde{J}$ then $T_{\tilde{\mu}}^{k+1}\tilde{J} \leq T_{\tilde{\mu}}^{k}\tilde{J}$ for all k, so, using also the fact $0 \leq \tilde{J}$, the limit of $T_{\tilde{\mu}}^{k}\tilde{J}$, call it J_∞, satisfies $J_{\tilde{\mu}} \leq J_\infty \leq \tilde{J}$). Thus if $\tilde{J}$ is nonnegative and real-valued, $J_{\tilde{\mu}}$ is also real-valued, so $\tilde{\mu}$ is stable. It follows that $\tilde{J}$ belongs to the region of stability. This is a known result in specific contexts, such as MPC (see the book by Rawlings, Mayne, and Diehl [RMD17], Section 2.4, which contains extensive references to prior work on stability issues).

An important special case where the condition $T\tilde{J} \leq \tilde{J}$ is satisfied is when $\tilde{J}$ is the cost function of a stable policy, i.e., $\tilde{J} = J_\mu$. Then we have that J_μ is real-valued and satisfies $T_\mu J_\mu = J_\mu$, so it follows that $TJ_\mu \leq J_\mu$. This case relates to the rollout algorithm and shows that *rollout with a*

stable policy yields a stable lookahead policy. It also suggests that if μ is stable, then $T_\mu^m \tilde{J}$ belongs to the region of stability for sufficiently large m.

Besides J_μ, with stable μ, and J^*, there are other interesting functions $\tilde{J}$ satisfying the stability condition $T\tilde{J} \leq \tilde{J}$. In particular, let β be a scalar with $\beta > 1$, and for a stable policy μ, consider the *β-amplified operator* $T_{\mu,\beta}$ defined by

$$(T_{\mu,\beta}J)(x) = E\Big\{\beta g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big)\Big\}, \qquad \text{for all } x.$$

Then it can be seen that the function

$$J_{\mu,\beta} = \beta J_\mu$$

is a fixed point of $T_{\mu,\beta}$ and satisfies $TJ_{\mu,\beta} \leq J_{\mu,\beta}$. This follows by writing

$$J_{\mu,\beta} = T_{\mu,\beta}J_{\mu,\beta} \geq T_\mu J_{\mu,\beta} \geq TJ_{\mu,\beta}. \tag{2.12}$$

Thus $J_{\mu,\beta}$ lies within the region of stability, and lies "further to the right" of J_μ. Thus we may conjecture that it can be more reliably approximated by $T_{\mu,\beta}^m \tilde{J}$ than J_μ is approximated by $T_\mu^m \tilde{J}$ in the context of m-step truncated rollout.

To illustrate this fact, consider a stable policy μ, and assume that the expected cost per stage at states other than a termination state t (if one exists) is bounded away from 0, i.e.,

$$C = \min_{x \neq t} E\Big\{g\big(x,\mu(x),w\big)\Big\} > 0.$$

Then we claim that given a scalar $\beta > 1$, any function $\hat{J} \in R(X)$ with $\hat{J}(t) = 0$, that satisfies

$$\max_x \big|\hat{J}(x) - J_{\mu,\beta}(x)\big| \leq \delta, \qquad \text{for all } x, \tag{2.13}$$

where

$$\delta = \frac{(\beta-1)C}{1+\alpha},$$

also satisfies the stability condition $T\hat{J} \leq \hat{J}$. From this it follows that *for a given nonnegative and real-valued $\tilde{J}$, and for sufficiently large m, so that the function $\hat{J} = T_{\mu,\beta}^m \tilde{J}$ satisfies Eq. (2.13), we have that $\hat{J}$ lies within the region of stability.*

To see this, note that for all $x \neq t$, we have

$$J_{\mu,\beta}(x) = \beta E\Big\{g\big(x,\mu(x),w\big)\Big\} + \alpha J_{\mu,\beta}\big(f(x,\mu(x),w)\big)\Big\},$$

so that by using Eq. (2.13), we have

$$\hat{J}(x) + \delta \geq \beta E\Big\{g\big(x,\mu(x),w\big)\Big\} + \alpha E\Big\{\hat{J}\big(f(x,\mu(x),w)\big)\Big\} - \alpha\delta.$$

It follows that

$$\begin{aligned}\hat{J}(x) &\geq E\Big\{g\big(x,\mu(x),w\big)\Big\} + \alpha E\Big\{\hat{J}\big(f(x,\mu(x),w)\big)\Big\} \\ &\qquad + (\beta-1)E\Big\{g\big(x,\mu(x),w\big)\Big\} - (1+\alpha)\delta \\ &\geq E\Big\{g\big(x,\mu(x),w\big)\Big\} + \alpha E\Big\{\hat{J}\big(f(x,\mu(x),w)\big)\Big\} + (\beta-1)C - (1+\alpha)\delta \\ &= (T_\mu \hat{J})(x) \\ &\geq (T\hat{J})(x),\end{aligned}$$

so the stability condition $T\hat{J} \leq \hat{J}$ is satisfied.

Similarly the function

$$J^*_\beta = \beta J^*$$

is a fixed point of the operator T_β defined by

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

It can be seen, using an argument similar to Eq. (2.12), that J^*_β satisfies $TJ^*_\beta \leq J^*_\beta$, so it lies within the region of stability. Furthermore, similar to the case of truncated rollout discussed earlier, we may conjecture that J^*_β can be more reliably approximated by $T^{\ell-1}_\beta \tilde{J}$ than J^* is approximated by $T^{\ell-1}\tilde{J}$ in the context of ℓ-step lookahead.

2.2.3 Policy Iteration, Rollout, and Newton's Method

Another major class of infinite horizon algorithms is based on *policy iteration* (PI for short), which involves the repeated use of policy improvement, in analogy with the AlphaZero/TD-Gammon off-line training algorithms, described in Section 1.1. Each iteration of the PI algorithm starts with a stable policy (which we call *current* or *base* policy), and generates another stable policy (which we call *new* or *rollout* policy, respectively). For the infinite horizon problem of Section 2.1, given the base policy μ, the iteration consists of two phases, as we have discussed in Chapter 1:

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

$$J_\mu(x) = E\Big\{g\big(x,\mu(x),w\big) + \alpha J_\mu\big(f(x,\mu(x),w)\big)\Big\}, \qquad \text{for all } x. \tag{2.14}$$

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 rollout policy $\tilde{\mu}$ using the one-step lookahead minimization

$$\tilde{\mu}(x) \in \arg\min_{u \in U(x)} E\Big\{g(x,u,w) + \alpha J_\mu\big(f(x,u,w)\big)\Big\}, \qquad \text{for all } x. \tag{2.15}$$

It is generally expected (and can be proved under mild conditions) that the rollout policy is improved in the sense that $J_{\tilde{\mu}}(x) \leq J_\mu(x)$ for all x.

Thus PI generates a sequence of stable policies $\{\mu^k\}$, by obtaining μ^{k+1} through a policy improvement operation using J_{μ^k} in place of J_μ in Eq. (2.15), which is obtained through policy evaluation of the preceding policy μ^k using Eq. (2.14). It is well known that (exact) PI has solid convergence properties; see the DP textbooks cited earlier, as well as the author's RL book [Ber19a]. These properties hold even when the method is implemented (with appropriate modifications) in unconventional computing environments, involving asynchronous distributed computation, as shown in a series of papers by Bertsekas and Yu [BeY10], [BeY12], [YuB13].

In terms of our abstract notation, the PI algorithm can be written in a compact form. For the generated policy sequence $\{\mu^k\}$, the policy evaluation phase obtains J_{μ^k} from the equation

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k}, \tag{2.16}$$

while the policy improvement phase obtains μ^{k+1} through the equation

$$T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}. \tag{2.17}$$

As Fig. 2.2.9 illustrates, PI can be viewed as Newton's method for solving the Bellman equation in the function space of cost functions J. In particular, *the policy improvement Eq. (2.17) is the Newton step starting from J_{μ^k}, and yields μ^{k+1} as the corresponding one-step lookahead/rollout policy*. Figure 2.2.10 illustrates the rollout algorithm, which is just the first iteration of PI.

In contrast to approximation in value space, the interpretation of PI in terms of Newton's method has a long history. We refer to the original works for linear quadratic problems by Kleinman [Klei68],† and for finite-state infinite horizon discounted and Markov game problems by Pollatschek

† This was part of Kleinman's Ph.D. thesis [Kle67] at M.I.T., supervised by M. Athans. Kleinman gives credit for the one-dimensional version of his results to Bellman and Kalaba [BeK65]. Note also that the first proposal of the PI method was given by Bellman in his classic book [Bel57], under the name "approximation in policy space."

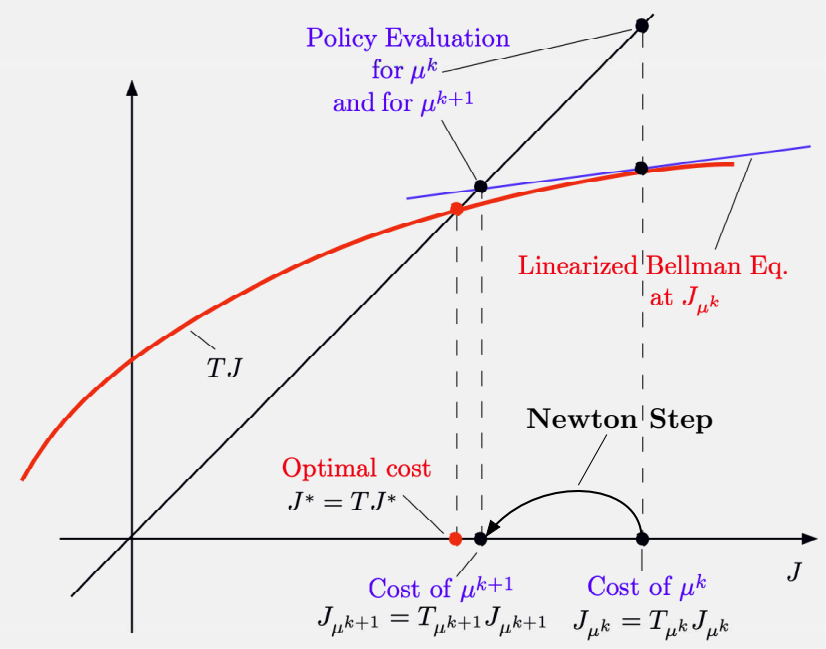

Figure 2.2.9 Geometric interpretation of a policy iteration. Starting from the stable current policy μ^k, it evaluates the corresponding cost function J_{μ^k}, and computes the next policy μ^{k+1} according to

$$T_{\mu^{k+1}} J_{\mu^k} = TJ_{\mu^k}.$$

The corresponding cost function $J_{\mu^{k+1}}$ is obtained as the solution of the linearized equation $J = T_{\mu^{k+1}} J$, so it is the result of a Newton step for solving the Bellman equation $J = TJ$, starting from J_{μ^k}. Note that in policy iteration, the Newton step always starts at a function J_μ, which satisfies $J_\mu \geq J^*$ as well as $TJ_\mu \leq J_\mu$ (cf. our discussion on stability in Section 3.2).

and Avi-Itzhak [PoA69] (who also showed that the method may oscillate in the game case). Subsequent works, which discuss algorithmic variations and approximations, include Hewer [Hew71], Puterman and Brumelle [PuB78], [PuB79], Saridis and Lee [SaL79] (following Rekasius [Rek64]), Beard [Bea95], Beard, Saridis, and Wen [BSW99], Santos and Rust [SaR04], Bokanowski, Maroso, and Zidani [BMZ09], Hylla [Hyl11], Magirou, Vassalos, and Barakitis [MVB20], Bertsekas [Ber21c], Ber22] and Kundu and Kunitsch [KuK21]. Some of these papers address broader classes of problems (such as continuous-time optimal control, minimax problems, and Markov games), include superlinear convergence rate results, as well as extensive references to related works. For a recent proof of quadratic convergence for linear discrete-time quadratic problems, see Lopez, Alsalti, and Muller

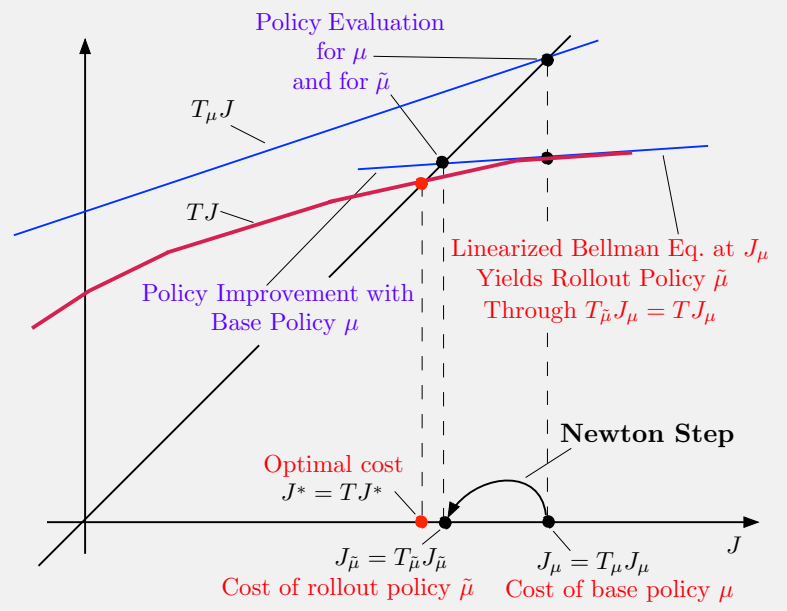

Figure 2.2.10 Geometric interpretation of rollout. Each policy μ defines the linear function $T_\mu J$ of J, given by Eq. (2.4), and TJ is the function given by Eq. (2.3), which can also be written as $TJ = \min_\mu T_\mu J$. The figure shows a policy iteration starting from a base policy μ. It computes J_μ by policy evaluation (by solving the linear equation $J = T_\mu J$ as shown). It then performs a policy improvement using μ as the base policy to produce the rollout policy $\tilde{\mu}$ as shown: the cost function of the rollout policy, $J_{\tilde{\mu}}$, is obtained by solving the version of Bellman's equation that is linearized at the point J_μ, as in Newton's method.

[LAM21].

Rollout

Generally, rollout with a stable base policy μ can be viewed as a single iteration of Newton's method starting from J_μ, as applied to the solution of the Bellman equation (see Fig. 2.2.10). Note that rollout/policy improvement is applied just at the current state during real-time operation of the system. This makes the on-line implementation possible, even for problems with very large state space, provided that the policy evaluation of the base policy can be done on-line as needed. For this we often need on-line deterministic or stochastic simulation from each of the states x_k generated by the system in real time.

As Fig. 2.2.10 illustrates, the cost function of the rollout policy $J_{\tilde{\mu}}$ is obtained by constructing a linearized version of Bellman's equation at J_μ (its linear approximation at J_μ), and then solving it. If the function TJ is nearly linear (i.e., has small "curvature") the rollout policy performance

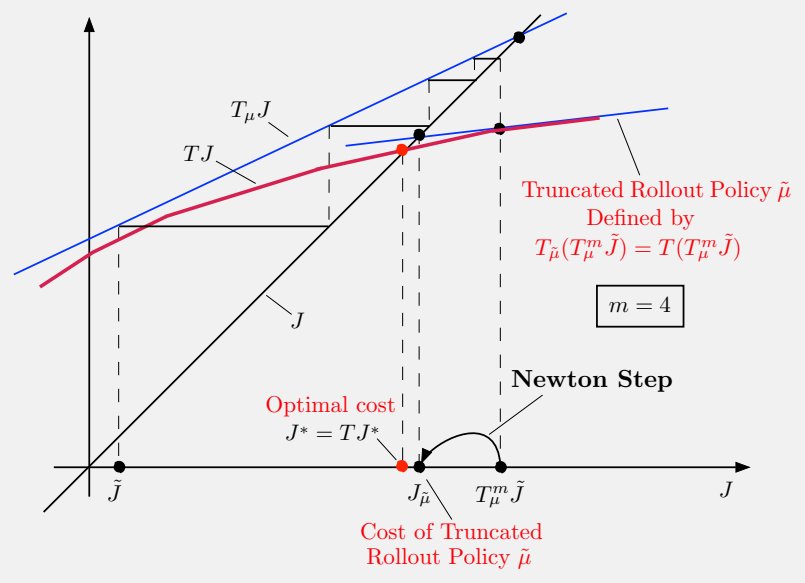

Figure 2.2.11 Geometric interpretation of truncated rollout with one-step lookahead minimization, m value iterations with the base policy μ, and a terminal cost function approximation $\tilde{J}$ (here $m = 4$).

$J_{\tilde{\mu}}(x)$ is very close to the optimal $J^*(x)$, even if the base policy μ is far from optimal. This explains the large cost improvements that are typically observed in practice with the rollout algorithm.

Truncated Rollout

Variants of rollout may involve multistep lookahead, truncation, and terminal cost function approximation, as in the case of AlphaZero/TD-Gammon, cf. Section 1. These variants admit geometric interpretations that are similar to the ones given earlier; see Fig. 2.2.11. Truncated rollout uses m VIs with the base policy μ and a terminal cost function approximation $\tilde{J}$ to approximate the cost function J_μ.

In the case of one-step lookahead, the truncated rollout policy $\tilde{\mu}$ is defined by

$$T_{\tilde{\mu}}(T_\mu^m \tilde{J}) = T(T_\mu^m \tilde{J}),$$

i.e., $\tilde{\mu}$ attains the minimum when the Bellman operator T is applied to the function $T_\mu^m \tilde{J}$ (the cost obtained by using the base policy μ for m steps followed by terminal cost approximation $\tilde{J}$); see Fig. 2.2.11. In the case of ℓ-step lookahead, the truncated rollout policy $\tilde{\mu}$ is defined by

$$T_{\tilde{\mu}}(T^{\ell-1} T_\mu^m \tilde{J}) = T(T^{\ell-1} T_\mu^m \tilde{J}).$$

Truncated rollout is related to a variant of PI called *optimistic*. This variant approximates the policy evaluation step by using m value iterations using the base policy μ; see [BeT96], [Ber12], [Ber19a] for a more detailed discussion of this relation.

As noted earlier, variants of Newton's method that involve multiple fixed point iterations, before and after each Newton step, but without truncated rollout, i.e.,

$$T_{\tilde{\mu}}(T^{\ell-1}\tilde{J}) = T(T^{\ell-1}\tilde{J}), \tag{2.18}$$

are well-known. The classical numerical analysis book by Ortega and Rheinboldt [OrR70] (Sections 13.3 and 13.4) provides various convergence results, under assumptions that include differentiability and convexity of the components of T, and nonnegativity of the inverse Jacobian of T. These assumptions, particularly differentiability, may not be satisfied within our DP context. Moreover, for methods of the form (2.18), the initial point must satisfy an additional assumption, which ensures that the convergence to J^* is monotonic from above (in this case, if in addition the Jacobian of T is isotone, an auxiliary sequence can be constructed that converges to J^* monotonically from below; see [OrR70], 13.3.4, 13.4.2). This is similar to existing convergence results for the optimistic PI method in DP; see e.g., [BeT96], [Ber12].

Geometrical interpretations such as the ones of Fig. 2.2.11 suggest, among others, that:

(a) The cost improvement $J_\mu - J_{\tilde{\mu}}$, from base to rollout policy, tends to become larger as the length ℓ of the lookahead increases.

(b) Truncated rollout with ℓ-step lookahead minimization, followed by m steps of a base policy μ, and then followed by terminal cost function approximation $\tilde{J}$ may be viewed, under certain conditions, as an economic alternative to $(\ell + m)$-step lookahead minimization using $\tilde{J}$ as terminal cost function approximation.

We next discuss the issues of selection of ℓ and m.

Lookahead Length Issues in Truncated Rollout

A question of practical interest is how to choose the lookahead lengths ℓ and m in truncated rollout schemes. It is clear that large values ℓ for lookahead minimization are beneficial (in the sense of producing improved lookahead policy cost functions $J_{\tilde{\mu}}$), since additional VI iterations bring closer to J^* the starting point $T^{\ell-1}\tilde{J}$ of the Newton step. On the other hand, large values m for truncated rollout bring the starting point for the Newton step closer to J_μ, and not necessarily closer to J^*. Indeed computational experiments suggest that *increasing values for m may be counterproductive beyond some threshold*, and that this threshold generally depends on the

problem and the terminal cost approximation $\tilde{J}$. This is also consistent with long standing experience with optimistic policy iteration, which is closely connected with truncated rollout, as noted earlier. Unfortunately, however, there is no analysis that can illuminate this issue, and the available error bounds for truncated rollout (see [Ber19a], [Ber20a]) are conservative and provide limited guidance in this regard.

An interesting property, which holds in some generality, is that *truncated rollout with a stable policy has a beneficial effect on the stability properties of the lookahead policy*. The reason is that the cost function J_μ of the base policy μ lies well inside the region of stability, as noted in Section 3.2. Moreover value iterations with μ (i.e., truncated rollout) tend to push the starting point of the Newton step towards J_μ. Thus a sufficient number of these value iterations will bring the starting point of the Newton step within the region of stability.

Another important fact to keep in mind is that the truncated rollout steps are much less demanding computationally than the lookahead minimization steps. In particular, large values of m may be computationally tolerable, but even relatively small values of m can be computationally prohibitive. This is especially true for stochastic problems where the width of the lookahead tree tends to grow quickly. Thus, with other concerns weighing equally, it is computationally preferable to use large values of m rather than large values of ℓ (this was the underlying motivation for truncated rollout in Tesauro's TD-Gammon [TeG96]).

The preceding discussion suggests the following qualitative question: *is lookahead by rollout an economic substitute for lookahead by minimization?* The answer to this seems to be a qualified yes: for a given computational budget, judiciously balancing the values of m and ℓ tends to give better lookahead policy performance than simply increasing ℓ as much as possible, while setting $m = 0$ (which corresponds to no rollout). This is consistent with intuition obtained through geometric constructions such as Fig. 2.2.11, but it is difficult to establish conclusively.

2.2.4 How Sensitive is On-Line Play to the Off-Line Training Process?

An important issue to consider in approximation in value space is errors in the one-step or multistep minimization, or in the choice of terminal cost approximation $\tilde{J}$. Such errors are often unavoidable because the control constraint set $U(x)$ is infinite, or because the minimization is simplified for reasons of computational expediency (see our subsequent discussion of multiagent problems). Moreover, to these errors, we may add the effect of errors due to rollout truncation, and errors due to changes in problem parameters, which are reflected in changes in Bellman's equation (see our subsequent discussion of robust and adaptive control).

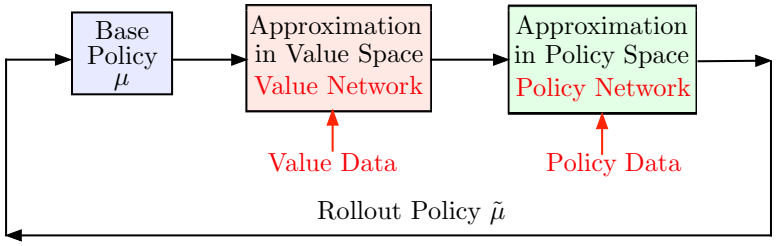

Figure 2.2.12 Schematic illustration of approximate PI. Either the policy evaluation and policy improvement phases (or both) are approximated with a value or a policy network, respectively. These could be neural networks, which are trained with (state, cost function value) data that is generated using the current base policy μ, and with (state, rollout policy control) data that is generated using the rollout policy $\tilde{\mu}$.

Note that there are three different types of approximate implementation involving: 1) a value network but no policy network (here the value network defines a policy via one-step or multistep lookahead), or 2) a policy network but no value network (here the policy network has a corresponding value function that can be computed by rollout), or 3) both a policy and a value network (the approximation architecture of AlphaZero is a case in point).

Under these circumstances the linearization of the Bellman equation at the point $\tilde{J}$ in Fig. 2.2.11 is perturbed, and the corresponding point $T_\mu^m \tilde{J}$ in Fig. 2.2.11 is also perturbed. However, the effect of these perturbations tends to be mitigated by the Newton step that produces the policy $\tilde{\mu}$ and the corresponding cost function $J_{\tilde{\mu}}$. The Newton step has a superlinear convergence property, so for an $O(\epsilon)$-order error [i.e., $O(\epsilon)/\epsilon$ stays bounded as $\epsilon \to 0$] in the calculation of $T_\mu^m \tilde{J}$, the error in $J_{\tilde{\mu}}$ will be of the much smaller order $o(\epsilon)$ [i.e., $o(\epsilon)/\epsilon \to 0$ as $\epsilon \to 0$], when $J_{\tilde{\mu}}$ is near J^*.† This is a significant insight, as it suggests that *extreme accuracy and fine-tuning of the choice of $\tilde{J}$ may not produce significant effects in the resulting performance of the one-step and particularly a multistep lookahead policy*; see also the discussion on linear quadratic problems in Section 1.5.

Approximate Policy Iteration and Implementation Errors

Both policy evaluation and policy improvement can be approximated, possibly by using training with data and approximation architectures, such as neural networks; see Fig. 2.2.12. Other approximations include simulation-based methods such as truncated rollout, and temporal difference methods for policy evaluation, which involve the use of basis functions. Moreover,

† A rigorous proof of this requires differentiability of T at $\tilde{J}$. Since T is differentiable at almost all points J, the sensitivity property just stated, will likely hold in practice even if T is not differentiable.

multistep lookahead may be used in place of one-step lookahead, and simplified minimization, based for example on multiagent rollout, may also be used. Let us also mention the possibility of a combined rollout and PI algorithm, whereby we use PI for on-line policy improvement of the base policy, by using data collected during the rollout process.

Long-standing practical experience with approximate PI is consistent with the view of the effect of implementation errors outlined above, and suggests that substantial changes in the policy evaluation and policy improvement operations often have small but largely unpredictable effects on the performance of the policies generated. For example, when TD(λ)-type methods are used for policy evaluation, the choice of λ has a large effect on the generated policy cost function approximations, but often has little and unpredictable effect on the performance of the generated policies. A plausible conjecture here is that the superlinear convergence property of the exact Newton step "smooths out" the effect of off-line approximation errors.

2.2.5 Why Not Just Train a Policy Network and Use it Without On-Line Play?

This is a sensible and common question, which stems from the mindset that neural networks have extraordinary function approximation properties. In other words, why go through the arduous on-line process of lookahead minimization, if we can do the same thing off-line and represent the lookahead policy with a trained policy network? More generally, it is possible to use *approximation in policy space*, a major alternative approach to approximation in value space, whereby we select the policy from a suitably restricted class of policies, such as a parametric class of the form $\mu(x, r)$, where r is a parameter vector. We may then estimate r using some type of off-line training process. There are quite a few methods for performing this type of training, such as policy gradient and random search methods (see the books [SuB18] and [Ber19a] for an overview). Alternatively, some approximate DP or classical control system design method may be used.

An important advantage of approximation in policy space is that once the parametrized policy is obtained, the on-line computation of controls $\mu(x, r)$ is often much faster compared with on-line lookahead minimization. For this reason, approximation in policy space can be used to provide an approximate implementation of a known policy (no matter how obtained) for the purpose of convenient use. On the negative side, because parametrized approximations often involve substantial calculations, they are not well suited for on-line replanning.

From our point of view in this book, there is another important reason why approximation in value space is needed on top of approximation in policy space: *the off-line trained policy may not perform nearly as well as the corresponding one-step or multistep lookahead/rollout policy*, because it

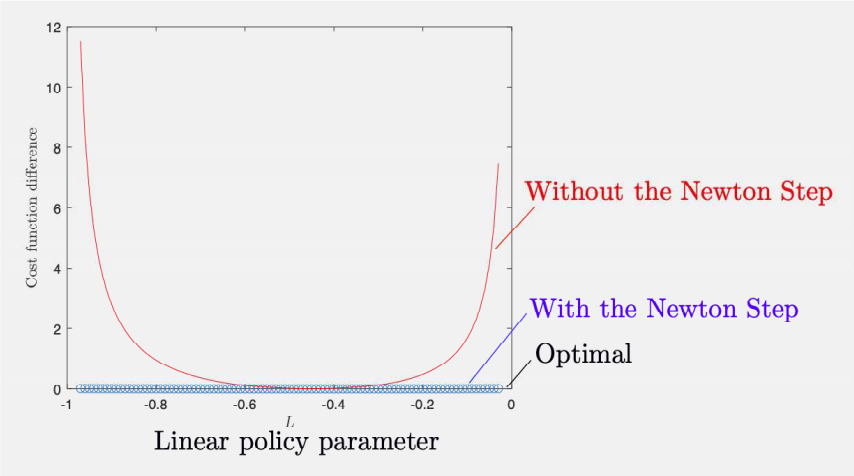

Figure 2.2.13 Illustration of the performance enhancement obtained by rollout with an off-line trained base policy for the linear quadratic problem. Here the system equation is $x_{k+1} = x_k + 2u_k$, and the cost function parameters are $q = 1$, $r = 0.5$. The optimal policy is $\mu^*(x) = L^*x$ with $L^* \approx -0.4$, and the optimal cost function is $J^*(x) = K^*x^2$, where $K^* \approx 1.1$. We consider policies of the form $\mu(x) = Lx$, where L is the parameter, with cost function of the form $J_\mu(x) = K_L x^2$. The figure shows the quadratic cost coefficient differences $K_L - K^*$ and $K_{\tilde{L}} - K^*$ as a function of L, where K_L and $K_{\tilde{L}}$ are the quadratic cost coefficients of μ (without one-step lookahead/Newton step) and the corresponding one-step lookahead policy $\tilde{\mu}$ (with one-step lookahead/Newton step).

lacks the extra power of the associated exact Newton step (cf. our discussion of AlphaZero and TD-Gammon in Section 1). Figure 2.2.13 illustrates this fact with a one-dimensional linear-quadratic example, and compares the performance of a linear policy, defined by a scalar parameter, with its corresponding one-step lookahead policy.

References

[ABB19] Agrawal, A., Barratt, S., Boyd, S., and Stellato, B., 2019. "Learning Convex Optimization Control Policies," arXiv preprint arXiv:1912.09529.

[ACF02] Auer, P., Cesa-Bianchi, N., and Fischer, P., 2002. "Finite Time Analysis of the Multiarmed Bandit Problem," Machine Learning, Vol. 47, pp. 235-256.

[ADH19] Arora, S., Du, S. S., Hu, W., Li, Z., and Wang, R., 2019. "Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks," arXiv preprint arXiv:1901.08584.

[AHZ19] Arcari, E., Hewing, L., and Zeilinger, M. N., 2019. "An Approximate Dynamic Programming Approach for Dual Stochastic Model Predictive Control," arXiv preprint arXiv:1911.03728.

[ALZ08] Asmuth, J., Littman, M. L., and Zinkov, R., 2008. "Potential-Based Shaping in Model-Based Reinforcement Learning," Proc. of 23rd AAAI Conference, pp. 604-609.

[AMS09] Audibert, J.Y., Munos, R., and Szepesvari, C., 2009. "Exploration-Exploitation Tradeoff Using Variance Estimates in Multi-Armed Bandits," Theoretical Computer Science, Vol. 410, pp. 1876-1902.

[ASR20] Andersen, A. R., Stidsen, T. J. R., and Reinhardt, L. B., 2020. "Simulation-Based Rolling Horizon Scheduling for Operating Theatres," in SN Operations Research Forum, Vol. 1, pp. 1-26.

[AXG16] Ames, A. D., Xu, X., Grizzle, J. W., and Tabuada, P., 2016. "Control Barrier Function Based Quadratic Programs for Safety Critical Systems," IEEE 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.

[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 preprint arXiv:2004.14427.

[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.

[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 preprint 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 preprint arXiv:1012.2599.

[BCN18] Bottou, L., Curtis, F. E., and Nocedal, J., 2018. "Optimization Methods for Large-Scale Machine Learning," SIAM Review, Vol. 60, pp. 223-311.

[BKB20] Bhattacharya, S., Kailas, S., Badyal, S., Gil, S., and Bertsekas, D. P., 2020. "Multiagent Rollout and Policy Iteration for POMDP with Application to Multi-Robot Repair Problems," in Proc. of Conference on Robot Learning (CoRL); also arXiv preprint, arXiv:2011.04222.

[BLL19] Bartlett, P. L., Long, P. M., Lugosi, G., and Tsigler, A., 2019. "Benign Overfitting in Linear Regression," arXiv preprint arXiv:1906.11300.

[BMM18] Belkin, M., Ma, S., and Mandal, S., 2018. "To Understand Deep Learning we Need to Understand Kernel Learning," arXiv preprint arXiv:1802.01396.

[BPW12] Browne, C., Powley, E., Whitehouse, D., Lucas, L., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S., 2012. "A Survey of Monte Carlo Tree Search Methods," IEEE Trans. on Computational Intelligence and AI in Games, Vol. 4, pp. 1-43.

[BRT18] Belkin, M., Rakhlin, A., and Tsybakov, A. B., 2018. "Does Data Interpolation Contradict Statistical Optimality?" arXiv preprint arXiv:1806.09471.

[BTW97] Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C., 1997. "Rollout Algorithms for Combinatorial Optimization," Heuristics, Vol. 3, pp. 245-262.

[BWL19] Beuchat, P. N., Warrington, J., and Lygeros, J., 2019. "Accelerated Point-Wise Maximum Approach to Approximate Dynamic Programming," arXiv preprint arXiv:1901.03619.

[BYB94] Bradtke, S. J., Ydstie, B. E., and Barto, A. G., 1994. "Adaptive Linear Quadratic Control Using Policy Iteration," Proc. IEEE American Control Conference, Vol. 3, pp. 3475-3479.

[BaF88] Bar-Shalom, Y., and Fortman, T. E., 1988. Tracking and Data Association, Academic Press, N. Y.

[BaL19] Banjac, G., and Lygeros, J., 2019. "A Data-Driven Policy Iteration Scheme Based on Linear Programming," Proc. 2019 IEEE CDC, pp. 816-821.

[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 Castanon, 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.

[Bel67] Bellman, R., 1967. Introduction to the Mathematical Theory of Control Processes, Academic Press, Vols. I and II, New York, N. Y.

[Bel84] Bellman, R., 1984. Eye of the Hurricane, World Scientific Publishing, Singapore.

[Ben09] Bengio, Y., 2009. "Learning Deep Architectures for AI," Foundations and Trends in Machine Learning, Vol. 2, pp. 1-127.

[Ber71] Bertsekas, D. P., 1971. "Control of Uncertain Systems With a Set-Membership Description of the Uncertainty," Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA (can be downloaded from the author's website).

[Ber72] Bertsekas, D. P., 1972. "Infinite Time Reachability of State Space Regions by Using Feedback Control," IEEE Trans. Automatic Control, Vol. AC-17, pp. 604-613.

[Ber73] Bertsekas, D. P., 1973. "Linear Convex Stochastic Control Problems over an Infinite Horizon," IEEE Trans. Automatic Control, Vol. AC-18, pp. 314-315.

[Ber77] Bertsekas, D. P., 1977. "Monotone Mappings with Application in Dynamic Programming," SIAM J. on Control and Opt., Vol. 15, pp. 438-464.

[Ber79] Bertsekas, D. P., 1979. "A Distributed Algorithm for the Assignment Problem," Lab. for Information and Decision Systems Report, 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.

[Ber11a] Bertsekas, D. P., 2011. "Incremental Proximal Methods for Large Scale Convex Optimization," Math. Programming, Vol. 129, pp. 163-195.

[Ber11b] Bertsekas, D. P., 2011. "Approximate Policy Iteration: A Survey and Some New Methods," J. of Control Theory and Applications, Vol. 9, pp. 310-335; a somewhat expanded version appears as Lab. for Info. and Decision Systems Report LIDS-2833, MIT, 2011.

[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 preprint arXiv:1507.1365936.

[Ber16] Bertsekas, D. P., 2016. Nonlinear Programming, 3rd Edition, Athena Scientific, Belmont, MA.

[Ber17] Bertsekas, D. P., 2017. Dynamic Programming and Optimal Control, Vol. I, 4th Edition, Athena Scientific, Belmont, MA.

[Ber18a] Bertsekas, D. P., 2018. Abstract Dynamic Programming, 2nd Edition, Athena Scientific, Belmont, MA (can be downloaded from the author's website).

[Ber18b] Bertsekas, D. P., 2018. "Feature-Based Aggregation and Deep Reinforcement Learning: A Survey and Some New Implementations," Lab. for Information and Decision Systems Report, MIT; arXiv preprint arXiv:1804.04577; IEEE/CAA Journal of Automatica Sinica, Vol. 6, 2019, pp. 1-31.

[Ber18c] Bertsekas, D. P., 2018. "Biased Aggregation, Rollout, and Enhanced Policy Improvement for Reinforcement Learning," Lab. for Information and Decision Systems Report, MIT; arXiv preprint arXiv:1910.02426.

[Ber18d] Bertsekas, D. P., 2018. "Proximal Algorithms and Temporal Difference Methods for Solving Fixed Point Problems," Computational Optim. Appl., Vol. 70, pp. 709-736.

[Ber19a] Bertsekas, D. P., 2019. Reinforcement Learning and Optimal Control, Athena Scientific, Belmont, MA.

[Ber19b] Bertsekas, D. P., 2019. "Robust Shortest Path Planning and Semicontractive Dynamic Programming," Naval Research Logistics, Vol. 66, pp. 15-37.

[Ber19c] Bertsekas, D. P., 2019. "Multiagent Rollout Algorithms and Reinforcement Learning," arXiv preprint arXiv:1910.00120.

[Ber19d] Bertsekas, D. P., 2019. "Constrained Multiagent Rollout and Multidimensional Assignment with the Auction Algorithm," arXiv preprint, arxiv.org/abs/2002.07407.

[Ber20a] Bertsekas, D. P., 2020. Rollout, Policy Iteration, and Distributed Reinforcement Learning, Athena Scientific, Belmont, MA.

[Ber20b] Bertsekas, D. P., 2020. "Multiagent Reinforcement Learning: Rollout and Policy Iteration," ASU Report; appears in IEEE/CAA Journal of Automatica Sinica, Vol. 8, 2021, pp. 249-271.

[Ber20c] 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.

[Ber21] Bertsekas, D. P., 2021. "Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control," arXiv preprint, arXiv:2108.10315.

[Ber22a] Bertsekas, D. P., 2022. Abstract Dynamic Programming, 3rd Edition, Athena Scientific, Belmont, MA (can be downloaded from the author's website).

[Ber22b] Bertsekas, D. P., 2022. Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control, Athena Scientific, Belmont, MA.

[Ber22c] Bertsekas, D. P., 2022. "Newton's Method for Reinforcement Learning and Model Predictive Control," ASU Report, January 2022; to appear in Results in Control and Optimization J.

[Bet10] Bethke, B. M., 2010. Kernel-Based Approximate Dynamic Programming Using Bellman Residual Elimination, Ph.D. Thesis, MIT.

[BiL97] Birge, J. R., and Louveaux, 1997. Introduction to Stochastic Programming, Springer, New York, N. Y.

[Bia16] Bianchi, P., 2016. "Ergodic Convergence of a Stochastic Proximal Point Algorithm," SIAM J. on Optimization, Vol. 26, pp. 2235-2260.

[Bis95] Bishop, C. M, 1995. Neural Networks for Pattern Recognition, Oxford University Press, N. Y.

[Bis06] Bishop, C. M, 2006. Pattern Recognition and Machine Learning, Springer, N. Y.

[BlG54] Blackwell, D., and Girshick, M. A., 1954. Theory of Games and Statistical Decisions, Wiley, N. Y.

[BlM08] Blanchini, F., and Miani, S., 2008. Set-Theoretic Methods in Control, Birkhauser, Boston.

[Bla86] Blackman, S. S., 1986. Multi-Target Tracking with Radar Applications, Artech House, Dehdam, MA.

[Bla99] Blanchini, F., 1999. "Set Invariance in Control – A Survey," Automatica, Vol. 35, pp. 1747-1768.

[Bod20] Bodson, M., 2020. Adaptive Estimation and Control, Independently Published.

[Bor08] Borkar, V. S., 2008. Stochastic Approximation: A Dynamical Systems Viewpoint, Cambridge Univ. Press.

[BrH75] Bryson, A., and Ho, Y. C., 1975. Applied Optimal Control: Optimization, Estimation, and Control, (revised edition), Taylor and Francis, Levittown, Penn.

[Bra21] Brandimarte, P., 2021. From Shortest Paths to Reinforcement Learning: A MATLAB-Based Tutorial on Dynamic Programming, Springer.

[BuK97] Burnetas, A. N., and Katehakis, M. N., 1997. "Optimal Adaptive Policies for Markov Decision Processes," Math. of Operations Research, Vol. 22, pp. 222-255.

[CBH09] Choi, H. L., Brunet, L., and How, J. P., 2009. "Consensus-Based Decentralized Auctions for Robust Task Allocation," IEEE 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.

[CLT19] Chapman, M. P., Lacotte, J., Tamar, A., Lee, D., Smith, K. M., Cheng, V., Fisac, J. F., Jha, S., Pavone, M., and Tomlin, C. J., 2019. "A Risk-Sensitive Finite-Time Reachability Approach for Safety of Stochastic Dynamic Systems," arXiv preprint arXiv:1902.11277.

[CMT87a] Clarke, D. W., Mohtadi, C., and Tuffs, P. S., 1987. "Generalized Predictive Control - Part I. The Basic Algorithm," Automatica Vol. 23, pp. 137-148.

[CMT87b] Clarke, D. W., Mohtadi, C., and Tuffs, P. S., 1987. "Generalized Predictive Control - Part II," Automatica Vol. 23, pp. 149-160.

[CRV06] Cogill, R., Rotkowitz, M., Van Roy, B., and Lall, S., 2006. "An Approximate Dynamic Programming Approach to Decentralized Control of Stochastic Systems," in Control of Uncertain Systems: Modelling, Approximation, and Design, Springer, Berlin, pp. 243-256.

[CXL19] Chu, Z., Xu, Z., and Li, H., 2019. "New Heuristics for the RCPSP with Multiple Overlapping Modes," Computers and Industrial Engineering, Vol. 131, pp. 146-156.

[CaB07] Camacho, E. F., and Bordons, C., 2007. Model Predictive Control, 2nd Edition, Springer, New York, N. Y.

[Can16] Candy, J. V., 2016. Bayesian Signal Processing: Classical, Modern, and Particle Filtering Methods, Wiley-IEEE Press.

[Cao07] Cao, X. R., 2007. Stochastic Learning and Optimization: A Sensitivity-Based Approach, Springer, N. Y.

[ChC17] Chui, C. K., and Chen, G., 2017. Kalman Filtering, Springer International Publishing.

[ChS00] Christianini, N., and Shawe-Taylor, J., 2000. Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge Univ. Press.

[Che59] Chernoff, H., 1959. "Sequential Design of Experiments," The Annals of Mathematical Statistics, Vol. 30, pp. 755-770.

[Chr97] Christodouleas, J. D., 1997. "Solution Methods for Multiprocessor Network Scheduling Problems with Application to Railroad Operations," Ph.D. Thesis, Operations Research Center, Massachusetts Institute of Technology.

[Cou06] Coulom, R., 2006. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," International Conference on Computers and Games, Springer, pp. 72-83.

[CrS00] Cristianini, N., and Shawe-Taylor, J., 2000. An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge Univ. Press.

[Cyb89] Cybenko, 1989. "Approximation by Superpositions of a Sigmoidal Function," Math. of Control, Signals, and Systems, Vol. 2, pp. 303-314.

[DDF19] Daubechies, I., DeVore, R., Foucart, S., Hanin, B., and Petrova, G., 2019. "Nonlinear Approximation and (Deep) ReLU Networks," arXiv preprint arXiv:1905.02199.

[DFM12] Desai, V. V., Farias, V. F., and Moallemi, C. C., 2012. "Aproximate Dynamic Programming via a Smoothed Approximate Linear Program," Operations Research, Vol. 60, pp. 655-674.

[DFM13] Desai, V. V., Farias, V. F., and Moallemi, C. C., 2013. "Bounds for Markov Decision Processes," in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, by F. Lewis and D. Liu (eds.), IEEE Press, Piscataway, N. J., pp. 452-473.

[DFV03] de Farias, D. P., and Van Roy, B., 2003. "The Linear Programming Approach to Approximate Dynamic Programming," Operations Research, Vol. 51, pp. 850-865.

[DFV04] de Farias, D. P., and Van Roy, B., 2004. "On Constraint Sampling in the Linear Programming Approach to Approximate Dynamic Programming," Mathematics of Operations Research, Vol. 29, pp. 462-478.

[DHS12] Duda, R. O., Hart, P. E., and Stork, D. G., 2012. Pattern Classification, J. Wiley, N. Y.

[DNW16] David, O. E., Netanyahu, N. S., and Wolf, L., 2016. "Deepchess: End-to-End Deep Neural Network for Automatic Learning in Chess," in International Conference on Artificial Neural Networks, pp. 88-96.

[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.

[DeK11] Devlin, S., and Kudenko, D., 2011. "Theoretical Considerations of Potential-Based Reward Shaping for Multi-Agent Systems," in Proceedings of AAMAS.

[Den67] Denardo, E. V., 1967. "Contraction Mappings in the Theory Underlying Dynamic Programming," SIAM Review, Vol. 9, pp. 165-177.

[DiL08] Dimitrakakis, C., and Lagoudakis, M. G., 2008. "Rollout Sampling Approximate Policy Iteration," Machine Learning, Vol. 72, pp. 157-171.

[DiM10] Di Castro, D., and Mannor, S., 2010. "Adaptive Bases for Reinforcement Learning," Machine Learning and Knowledge Discovery in Databases, Vol. 6321, pp. 312-327.

[DiW02] Dietterich, T. G., and Wang, X., 2002. "Batch Value Function Approximation via Support Vectors," in Advances in Neural Information Processing Systems, pp. 1491-1498.

[DoJ09] Doucet, A., and Johansen, A. M., 2009. "A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later," Handbook of Nonlinear Filtering, Oxford University Press, Vol. 12, p. 3.

[DrH01] Drezner, Z., and Hamacher, H. W. eds., 2001. Facility Location: Applications and Theory, Springer Science and Business Media.

[DuV99] Duin, C., and Voss, S., 1999. "The Pilot Method: A Strategy for Heuristic Repetition with Application to the Steiner Problem in Graphs," Networks: An International Journal, Vol. 34, pp. 181-191.

[EDS18] Efroni, Y., Dalal, G., Scherrer, B., and Mannor, S., 2018. "Beyond the One-Step Greedy Approach in Reinforcement Learning," in Proc. International Conf. on Machine Learning, pp. 1387-1396.

[EMM05] Engel, Y., Mannor, S., and Meir, R., 2005. "Reinforcement Learning with Gaussian Processes," in Proc. of the 22nd ICML, pp. 201-208.

[FHS09] Feitzinger, F., Hylla, T., and Sachs, E. W., 2009. "Inexact Kleinman?Newton Method for Riccati Equations," SIAM 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.

[FiV96] Filar, J., and Vrieze, K., 1996. Competitive Markov Decision Processes, Springer.

[FoK09] Forrester, A. I., and Keane, A. J., 2009. "Recent Advances in Surrogate-Based Optimization. Progress in Aerospace Sciences," Vol. 45, pp. 50-79.

[Fra18] Frazier, P. I., 2018. "A Tutorial on Bayesian Optimization," arXiv preprint arXiv:1807.02811.

[Fu17] Fu, M. C., 2017. "Markov Decision Processes, AlphaGo, and Monte Carlo Tree Search: Back to the Future," Leading Developments from INFORMS Communities, INFORMS, pp. 68-88.

[Fun89] Funahashi, K., 1989. "On the Approximate Realization of Continuous Mappings by Neural Networks," Neural Networks, Vol. 2, pp. 183-192.

[GBC16] Goodfellow, I., Bengio, J., and Courville, A., Deep Learning, MIT Press, Cambridge, MA.

[GBL19] Goodson, J. C., Bertazzi, L., and Levary, R. R., 2019. "Robust Dynamic Media Selection with Yield Uncertainty: Max-Min Policies and Dual Bounds," Report.

[GDM19] Guerriero, F., Di Puglia Pugliese, L., and Macrina, G., 2019. "A Rollout Algorithm for the Resource Constrained Elementary Shortest Path Problem," Optimization Methods and Software, Vol. 34, pp. 1056-1074.

[GGS13] Gabillon, V., Ghavamzadeh, M., and Scherrer, B., 2013. "Approximate Dynamic Programming Finally Performs Well in the Game of Tetris," in NIPS, pp. 1754-1762.

[GGW11] Gittins, J., Glazebrook, K., and Weber, R., 2011. Multi-Armed Bandit Allocation Indices, J. Wiley, N. Y.

[GLG11] Gabillon, V., Lazaric, A., Ghavamzadeh, M., and Scherrer, B., 2011. "Classification-Based Policy Iteration with a Critic," in Proc. of ICML.

[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.

[GoS84] Goodwin, G. C., and Sin, K. S. S., 1984. Adaptive Filtering, Prediction, and Control, Prentice-Hall, Englewood Cliffs, N. J.

[Gos15] Gosavi, A., 2015. Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning, 2nd 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 preprint 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.

[HaR21] Hardt, M., and Recht, B., 2021. Patterns, Predictions, and Actions: A Story About Machine Learning, arXiv preprint arXiv:2102.05242.

[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 preprint 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 preprint arXiv:1910.00614.

[IJT18] Iusem, Jofre, A., and Thompson, P., 2018. "Incremental Constraint Projection Methods for Monotone Stochastic Variational Inequalities," Math. of Operations Research, Vol. 44, pp. 236-263.

[IoS96] Ioannou, P. A., and Sun, J., 1996. Robust Adaptive Control, Prentice-Hall, Englewood Cliffs, N. J.

[JCG20] Jiang, S., Chai, H., Gonzalez, J., and Garnett, R., 2020. "BINOCULARS for Efficient, Nonmyopic Sequential Experimental Design," in Proc. Intern. Conference on Machine Learning, pp. 4794-4803.

[JGJ18] Jones, M., Goldstein, M., Jonathan, P., and Randell, D., 2018. "Bayes Linear Analysis of Risks in Sequential Optimal Design Problems," Electronic 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.

[JiJ17] Jiang, Y., and Jiang, Z. P., 2017. Robust Adaptive Dynamic Programming, J. Wiley, N. Y.

[Jon90] Jones, L. K., 1990. "Constructive Approximations for Neural Networks by Sigmoidal Functions," Proceedings of the IEEE, Vol. 78, pp. 1586-1589.

[JuP07] Jung, T., and Polani, D., 2007. "Kernelizing LSPE(λ)," Proc. 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning, Honolulu, Ha., pp. 338-345.

[KAC15] Kochenderfer, M. J., with Amato, C., Chowdhary, G., How, J. P., Davison Reynolds, H. J., Thornton, J. R., Torres-Carrasquillo, P. A., Ore, N. K., Vian, J., 2015. Decision Making under Uncertainty: Theory and Application, MIT Press, Cambridge, MA.

[KAH15] Khashooei, B. A., Antunes, D. J. and Heemels, W.P.M.H., 2015. "Rollout Strategies for Output-Based Event-Triggered Control," in Proc. 2015 European Control Conference, pp. 2168-2173.

[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.

[KaW94] Kall, P., and Wallace, S. W., 1994. Stochastic Programming, Wiley, Chichester, UK.

[KeG88] Keerthi, S. S., and Gilbert, E. G., 1988. "Optimal, Infinite Horizon Feedback Laws for a General Class of Constrained Discrete Time Systems: Stability and Moving-Horizon Approximations," J. Optimization Theory Appl., Vo. 57, pp. 265-293.

[Kle68] Kleinman, D. L., 1968. "On an Iterative Technique for Riccati Equation Computations," IEEE Trans. Aut. Control, Vol. AC-13, pp. 114-115.

[KoC16] Kouvaritakis, B., and Cannon, M., 2016. Model Predictive Control: Classical, Robust and Stochastic, Springer, N. Y.

[KoG98] Kolmanovsky, I.. and Gilbert, E. G., 1998. "Theory and Computation of Disturbance Invariant Sets for Discrete-Time Linear Systems," Math. Problems in Engineering, Vol. 4, pp. 317-367.

[KoS06] Kocsis, L., and Szepesvari, C., 2006. "Bandit Based Monte-Carlo Planning," Proc. of 17th European Conference on Machine Learning, Berlin, pp. 282-293.

[Kre19] Krener, A. J., 2019. "Adaptive Horizon Model Predictive Control and Al'brekht's Method," arXiv preprint arXiv:1904.00053.

[Kri16] Krishnamurthy, V., 2016. Partially Observed Markov Decision Processes, Cambridge Univ. Press.

[KuV86] Kumar, P. R., and Varaiya, P. P., 1986. Stochastic Systems: Estimation, Identification, and Adaptive Control, Prentice-Hall, Englewood Cliffs, N. J.

[Kun14] Kung, S. Y., 2014. Kernel Methods and Machine Learning, Cambridge Univ. Press.

[LEC20] Lee, E. H., Eriksson, D., Cheng, B., McCourt, M., and Bindel, D., 2020. "Efficient Rollout Strategies for Bayesian Optimization," arXiv preprint arXiv:2002.10539.

[LGM10] Lazaric, A., Ghavamzadeh, M., and Munos, R., 2010. "Analysis of a Classification-Based Policy Iteration Algorithm," INRIA Report.

[LGW16] Lan, Y., Guan, X., and Wu, J., 2016. "Rollout Strategies for Real-Time Multi-Energy Scheduling in Microgrid with Storage System," IET Generation, Transmission and Distribution, Vol. 10, pp. 688-696.

[LJM19] Li, Y., Johansson, K. H., and Martensson, J., 2019. "Lambda-Policy Iteration with Randomization for Contractive Models with Infinite Policies: Well Posedness and Convergence," arXiv preprint arXiv:1912.08504.

[LLL19] Liu, Z., Lu, J., Liu, Z., Liao, G., Zhang, H. H., and Dong, J., 2019. "Patient Scheduling in Hemodialysis Service," J. of Combinatorial Optimization, Vol. 37, pp. 337-362.

[LLP93] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S., 1993. "Multilayer Feedforward Networks with a Nonpolynomial Activation Function can Approximate any

Function," Neural Networks, Vol. 6, pp. 861-867.

[LPS21] Liu, M., Pedrielli, G., Sulc, P., Poppleton, E., Bertsekas, D. P., 2021. "ExpertRNA: A New Framework for RNA Structure Prediction," bioRxiv.

[LTZ19] Li, Y., Tang, Y., Zhang, R., and Li, N., 2019. "Distributed Reinforcement Learning for Decentralized Linear Quadratic Control: A Derivative-Free Policy Optimization Approach," arXiv preprint arXiv:1912.09135.

[LWT17] Lowe, L., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I., 2017. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," in Advances in Neural Information Processing Systems, pp. 6379-6390.

[LWW16] Lam, R., Willcox, K., and Wolpert, D. H., 2016. "Bayesian Optimization with a Finite Budget: An Approximate Dynamic Programming Approach," In Advances in Neural Information Processing Systems, pp. 883-891.

[LWW17] Liu, D., Wei, Q., Wang, D., Yang, X., and Li, H., 2017. Adaptive Dynamic Programming with Applications in Optimal Control, Springer, Berlin.

[LZS20] Li, H., Zhang, X., Sun, J., and Dong, X., 2020. "Dynamic Resource Levelling in Projects under Uncertainty," International J. of Production Research.

[LaP03] Lagoudakis, M. G., and Parr, R., 2003. "Reinforcement Learning as Classification: Leveraging Modern Classifiers," in Proc. of ICML, pp. 424-431.

[LaR85] Lai, T., and Robbins, H., 1985. "Asymptotically Efficient Adaptive Allocation Rules," Advances in Applied Math., Vol. 6, pp. 4-22.

[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.

[LiS16] Liang, S., and Srikant, R., 2016. "Why Deep Neural Networks for Function Approximation?" arXiv preprint arXiv:1610.04161.

[LiW14] Liu, D., and Wei, Q., 2014. "Policy Iteration Adaptive Dynamic Programming Algorithm for Discrete-Time Nonlinear Systems," IEEE Trans. on Neural Networks and Learning Systems, Vol. 25, pp. 621-634.

[LiW15] Li, H., and Womer, N. K., 2015. "Solving Stochastic Resource-Constrained Project Scheduling Problems by Closed-Loop Approximate Dynamic Programming," European J. of Operational Research, Vol. 246, pp. 20-33.

[Lib11] Liberzon, D., 2011. Calculus of Variations and Optimal Control Theory: A Concise Introduction, Princeton Univ. Press.

[KKK95] Krstic, M., Kanellakopoulos, I., Kokotovic, P., 1995. Nonlinear and Adaptive Control Design, J. Wiley, N. Y.

[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.

[MLM20] Montenegro, M., Lopez, R., Menchaca-Mendez, R., Becerra, E., and Menchaca-Mendez, R., 2020. "A Parallel Rollout Algorithm for Wildfire Suppression," in Proc. Intern. Congress of Telematics and Computing, pp. 244-255.

[MMB02] McGovern, A., Moss, E., and Barto, A., 2002. "Building a Basic Building Block Scheduler Using Reinforcement Learning and Rollouts," Machine Learning, Vol. 49, pp. 141-160.

[MMS05] Menache, I., Mannor, S., and Shimkin, N., 2005. "Basis Function Adaptation in Temporal Difference Reinforcement Learning," Ann. Oper. Res., Vol. 134, pp. 215-238.

[MPK99] Meuleau, N., Peshkin, L., Kim, K. E., and Kaelbling, L. P., 1999. "Learning Finite-State Controllers for Partially Observable Environments," in Proc. of the 15th Conference on Uncertainty in Artificial Intelligence, pp. 427-436.

[MPP04] Meloni, C., Pacciarelli, D., and Pranzo, M., 2004. "A Rollout Metaheuristic for Job Shop Scheduling Problems," Annals of Operations Research, Vol. 131, pp. 215-235.

[MRR00] Mayne, D., Rawlings, J. B., Rao, C. V., and Scokaert, P. O. M., 2000. "Constrained Model Predictive Control: Stability and Optimality," Automatica, Vol. 36, pp. 789-814.

[MVS19] Muthukumar, V., Vodrahalli, K., and Sahai, A., 2019. "Harmless Interpolation of Noisy Data in Regression," arXiv preprint arXiv:1903.09139.

[MYF03] Moriyama, H., Yamashita, N., and Fukushima, M., 2003. "The Incremental Gauss-Newton Algorithm with Adaptive Stepsize Rule," Computational Optimization and Applications, Vol. 26, pp. 107-141.

[MaJ15] Mastin, A., and Jaillet, P., 2015. "Average-Case Performance of Rollout Algorithms for Knapsack Problems," J. of Optimization Theory and Applications, Vol. 165, pp. 964-984.

[Mac02] Maciejowski, J. M., 2002. Predictive Control with Constraints, Addison-Wesley, Reading, MA.

[Mar55] Marschak, J., 1975. "Elements for a Theory of Teams," Management Science, Vol. 1, pp. 127-137.

[Mar84] Martins, E. Q. V., 1984. "On a Multicriteria Shortest Path Problem," European J. of Operational Research, Vol. 16, pp. 236-245.

[May14] Mayne, D. Q., 2014. "Model Predictive Control: Recent Developments and Future Promise," Automatica, Vol. 50, pp. 2967-2986.

[MeB99] Meuleau, N., and Bourgine, P., 1999. "Exploration of Multi-State Environments: Local Measures and Back-Propagation of Uncertainty," Machine Learning, Vol. 35, pp. 117-154.

[MeK20] Meshram, R.. and Kaza, K., 2020. "Simulation Based Algorithms for Markov Decision Processes and Multi-Action Restless Bandits," arXiv preprint arXiv:2007.12933.

[Mey07] Meyn, S., 2007. Control Techniques for Complex Networks, Cambridge Univ. Press, N. Y.

[Min22] Minorsky, N., 1922. "Directional Stability of Automatically Steered Bodies," J. Amer. Soc. Naval Eng.,Vol. 34, pp. 280-309.

[MoL99] Morari, M., and Lee, J. H., 1999. "Model Predictive Control: Past, Present, and Future," Computers and Chemical Engineering, Vol. 23, pp. 667-682.

[Mon17] Montgomery, D. C., 2017. Design and Analysis of Experiments, J. Wiley.

[Mun14] Munos, R., 2014. "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning," Foundations and Trends in Machine Learning, Vol. 7, pp. 1-129.

[NHR99] Ng, A. Y., Harada, D., and Russell, S. J., 1999. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping," in Proc. of the 16th International Conference on Machine Learning, pp. 278-287.

[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.

[NaA12] Narendra, K. S., and Annaswamy, A. M., 2012. Stable Adaptive Systems, Courier Corporation.

[NaT19] Nayyar, A., and Teneketzis, D., 2019. "Common Knowledge and Sequential Team Problems," IEEE Trans. on Automatic Control, Vol. 64, pp. 5108-5115.

[Ned11] Nedić, A., 2011. "Random Algorithms for Convex Minimization Problems," Math. Programming, Ser. B, Vol. 129, pp. 225-253.

[OrS02] Ormoneit, D., and Sen, S., 2002. "Kernel-Based Reinforcement Learning," Machine Learning, Vol. 49, pp. 161-178.

[PDB92] Pattipati, K. R., Deb, S., Bar-Shalom, Y., and Washburn, R. B., 1992. "A New Relaxation Algorithm and Passive Sensor Data Association," IEEE Trans. Automatic Control, Vol. 37, pp. 198-213.

[PDC14] Pillonetto, G., Dinuzzo, F., Chen, T., De Nicolao, G., and Ljung, L., 2014. "Kernel Methods in System Identification, Machine Learning and Function Estimation: A Survey," Automatica, Vol. 50, pp. 657-682.

[PPB01] Popp, R. L., Pattipati, K. R., and Bar-Shalom, Y., 2001. "m-Best SD Assignment Algorithm with Application to Multitarget Tracking," IEEE Transactions on Aerospace and Electronic Systems, Vol. 37, pp. 22-39.

[PaB99] Patek, S. D., and Bertsekas, D. P., 1999. "Stochastic Shortest Path Games," SIAM J. on Control and Optimization, Vol. 37, pp. 804-824.

[PaR12] Papahristou, N., and Refanidis, I., 2012. "On the Design and Training of Bots to Play Backgammon Variants," in IFIP International Conference on Artificial Intelligence Applications and Innovations, pp. 78-87.

[PaT00] Paschalidis, I. C., and Tsitsiklis, J. N., 2000. "Congestion-Dependent Pricing of Network Services," IEEE/ACM Trans. on Networking, Vol. 8, pp. 171-184.

[PeG04] Peret, L., and Garcia, F., 2004. "On-Line Search for Solving Markov Decision Processes via Heuristic Sampling," in Proc. of the 16th European Conference on Artificial Intelligence, pp. 530-534.

[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.

[Pre95] Prekopa, A., 1995. Stochastic Programming, Kluwer, Boston.

[PuB78] Puterman, M. L., and Brumelle, S. L., 1978. "The Analytic Theory of Policy Iteration," in Dynamic Programming and Its Applications, M. L. Puterman (ed.), Academic Press, N. Y.

[PuB79] Puterman, M. L., and Brumelle, S. L., 1979. "On the Convergence of Policy Iteration in Stationary Dynamic Programming," Mathematics of Operations Research, Vol. 4, pp. 60-69.

[PuS78] Puterman, M. L., and Shin, M. C., 1978. "Modified Policy Iteration Algorithms for Discounted Markov Decision Problems," Management Sci., Vol. 24, pp. 1127-1137.

[PuS82] Puterman, M. L., and Shin, M. C., 1982. "Action Elimination Procedures for Modified Policy Iteration Algorithms," Operations Research, Vol. 30, pp. 301-318.

[Put94] Puterman, M. L., 1994. Markovian Decision Problems, J. Wiley, N. Y.

[QHS05] Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R., and Tucker, P. K., 2005. "Surrogate-Based Analysis and Optimization," Progress in Aerospace Sciences, Vol. 41, pp. 1-28.

[QuL19] Qu, G., and Li, N., "Exploiting Fast Decaying and Locality in Multi-Agent MDP with Tree Dependence Structure," Proc. of 2019 CDC, Nice, France.

[RCR17] Rudi, A., Carratino, L., and Rosasco, L., 2017. "Falkon: An Optimal Large Scale Kernel Method," in Advances in Neural Information Processing Systems, pp. 3888-3898.

[RGG21] Rimélé, A., Grangier, P., Gamache, M., Gendreau, M., and Rousseau, L. M., 2021. "E-Commerce Warehousing: Learning a Storage Policy, arXiv:2101.08828.

[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.

[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.

[RST20] Rusmevichientong, P., Sumida, M., Topaloglu, H., and Bai, Y., 2020. "Revenue Management for Boutique Hotels: Resources with Unit Capacities and Itineraries over Intervals of Resources," Cornell Univ. Report.

[RaF91] Raghavan, T. E. S., and Filar, J. A., 1991. "Algorithms for Stochastic Games - A Survey," Zeitschrift fur Operations Research, Vol. 35, pp. 437-472.

[RaR17] Rawlings, J. B., and Risbeck, M. J., 2017. "Model Predictive Control with Discrete Actuators: Theory and Application," Automatica, Vol. 78, pp. 258-265.

[RaW06] Rasmussen, C. E., and Williams, C. K., 2006. Gaussian Processes for Machine Learning, MIT Press, Cambridge, MA.

[Rad62] Radner, R., 1962. "Team Decision Problems," Ann. Math. Statist., Vol. 33, pp. 857-881.

[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.

[RuK16] Rubinstein, R. Y., and Kroese, D. P., 2016. Simulation and the Monte Carlo Method, 3rd Edition, J. Wiley, N. Y.

[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.

[SGC02] Savagaonkar, U., Givan, R., and Chong, E. K. P., 2002. "Sampling Techniques for Zero-Sum, Discounted Markov Games," in Proc. 40th Allerton Conference on Communication, Control and Computing, Monticello, Ill.

[SGG15] Scherrer, B., Ghavamzadeh, M., Gabillon, V., Lesner, B., and Geist, M., 2015. "Approximate Modified Policy Iteration and its Application to the Game of Tetris," J. of Machine Learning Research, Vol. 16, pp. 1629-1676.

[SHB15] Simroth, A., Holfeld, D., and Brunsch, R., 2015. "Job Shop Production Planning under Uncertainty: A Monte Carlo Rollout Approach," Proc. of the International Scientific and Practical Conference, Vol. 3, pp. 175-179.

[SHM16] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., and Dieleman, S., 2016. "Mastering the Game of Go with Deep Neural Networks and Tree Search," Nature, Vol. 529, pp. 484-489.

[SHS17] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., and Lillicrap, T., 2017. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," arXiv preprint arXiv:1712.01815.

[SJL18] Soltanolkotabi, M., Javanmard, A., and Lee, J. D., 2018. "Theoretical Insights into the Optimization Landscape of Over-Parameterized Shallow Neural Networks," IEEE Trans. on Information Theory, Vol. 65, pp. 742-769.

[SLA12] Snoek, J., Larochelle, H., and Adams, R. P., 2012. "Practical Bayesian Optimization of Machine Learning Algorithms," in Advances in Neural Information Processing Systems, pp. 2951-2959.

[SLJ13] Sun, B., Luh, P. B., Jia, Q. S., Jiang, Z., Wang, F., and Song, C., 2013. "Building Energy Management: Integrated Control of Active and Passive Heating, Cooling, Lighting, Shading, and Ventilation Systems," IEEE Trans. on Automation Science and Engineering, Vol. 10, pp. 588-602.

[SNC18] Sarkale, Y., Nozhati, S., Chong, E. K., Ellingwood, B. R., and Mahmoud, H., 2018. "Solving Markov Decision Processes for Network-Level Post-Hazard Recovery via Simulation Optimization and Rollout," in 2018 IEEE 14th International Conference on Automation Science and Engineering, pp. 906-912.

[SSS17] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. and Chen, Y., 2017. "Mastering the Game of Go Without Human Knowledge," Nature, Vol. 550, pp. 354-359.

[SSW16] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N., 2015. "Taking the Human Out of the Loop: A Review of Bayesian Optimization," Proc. of IEEE, Vol. 104, pp. 148-175.

[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.

[SYL17] Saldi, N., Yuksel, S., and Linder, T., 2017. "Finite Model Approximations for Partially Observed Markov Decision Processes with Discounted Cost," arXiv preprint arXiv:1710.07009.

[SZL08] Sun, T., Zhao, Q., Lun, P., and Tomastik, R., 2008. "Optimization of Joint Replacement Policies for Multipart Systems by a Rollout Framework," IEEE Trans. on Automation Science and Engineering, Vol. 5, pp. 609-619.

[SaB11] Sastry, S., and Bodson, M., 2011. Adaptive Control: Stability, Convergence and Robustness, Courier Corporation.

[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.

[SiK19] Singh, R., and Kumar, P. R., 2019. "Optimal Decentralized Dynamic Policies for Video Streaming over Wireless Channels," arXiv preprint arXiv:1902.07418.

[SlL91] Slotine, J.-J. E., and Li, W., Applied Nonlinear Control, Prentice-Hall, Englewood Cliffs, N. J.

[StW91] Stewart, B. S., and White, C. C., 1991. "Multiobjective A^*," J. ACM, Vol. 38, pp. 775-814.

[SuB18] Sutton, R., and Barto, A. G., 2018. Reinforcement Learning, 2nd 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 preprint arXiv:1912.08957.

[Sze10] Szepesvari, C., 2010. Algorithms for Reinforcement Learning, Morgan and Claypool Publishers, San Franscisco, CA.

[TBP21] Tuncel, Y., Bhat, G., Park, J., and Ogras, U., 2021. "ECO: Enabling Energy-Neutral IoT Devices through Runtime Allocation of Harvested Energy," arXiv preprint arXiv:2102.13605.

[TCW19] Tseng, W. J., Chen, J. C., Wu, I. C., and Wei, T. H., 2019. "Comparison Training for Computer Chinese Chess," IEEE Trans. on Games, Vol. 12, pp. 169-176.

[TGL13] Tesauro, G., Gondek, D. C., Lenchner, J., Fan, J., and Prager, J. M., 2013. "Analysis of Watson's Strategies for Playing Jeopardy!," J. of Artificial Intelligence Research, Vol. 47, pp. 205-251.

[TRV16] Tu, S., Roelofs, R., Venkataraman, S., and Recht, B., 2016. "Large Scale Kernel Learning Using Block Coordinate Descent," arXiv preprint arXiv:1602.05310.

[TaL20] Tanzanakis, A., and Lygeros, J., 2020. "Data-Driven Control of Unknown Systems: A Linear Programming Approach," arXiv preprint arXiv:2003.00779.

[TeG96] Tesauro, G., and Galperin, G. R., 1996. "On-Line Policy Improvement Using Monte Carlo Search," NIPS, Denver, CO.

[Tes89a] Tesauro, G. J., 1989. "Neurogammon Wins Computer Olympiad," Neural Computation, Vol. 1, pp. 321-323.

[Tes89b] Tesauro, G. J., 1989. "Connectionist Learning of Expert Preferences by Comparison Training," in Advances in Neural Information Processing Systems, pp. 99-106.

[Tes92] Tesauro, G. J., 1992. "Practical Issues in Temporal Difference Learning," Machine Learning, Vol. 8, pp. 257-277.

[Tes94] Tesauro, G. J., 1994. "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play," Neural Computation, Vol. 6, pp. 215-219.

[Tes95] Tesauro, G. J., 1995. "Temporal Difference Learning and TD-Gammon," Communications of the ACM, Vol. 38, pp. 58-68.

[Tes01] Tesauro, G. J., 2001. "Comparison Training of Chess Evaluation Functions," in Machines that Learn to Play Games, Nova Science Publishers, pp. 117-130.

[Tes02] Tesauro, G. J., 2002. "Programming Backgammon Using Self-Teaching Neural Nets," Artificial Intelligence, Vol. 134, pp. 181-199.

[ThS09] Thiery, C., and Scherrer, B., 2009. "Improvements on Learning Tetris with Cross-Entropy," International Computer Games Association J., Vol. 32, pp. 23-33.

[TsV96] Tsitsiklis, J. N., and Van Roy, B., 1996. "Feature-Based Methods for Large-Scale Dynamic Programming," Machine Learning, Vol. 22, pp. 59-94.

[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.

[VPA09] Vrabie, D., Pastravanu, O., Abu-Khalaf, M., and Lewis, F. L., 2009. "Adaptive Optimal Control for Continuous-Time Linear Systems Based on Policy Iteration," Automatica, Vol. 45, pp. 477-484.

[VVL13] Vrabie, D., Vamvoudakis, K. G., and Lewis, F. L., 2013. Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles, The Institution of Engineering and Technology, London.

[Van76] Van Nunen, J. A., 1976. Contracting Markov Decision Processes, Mathematical Centre Report, Amsterdam.

[WCG02] Wu, G., Chong, E. K. P., and Givan, R. L., 2002. "Burst-Level Congestion Control Using Hindsight Optimization," IEEE 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.

[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.

[WaS00] de Waal, P. R., and van Schuppen, J. H., 2000. "A Class of Team Problems with Discrete Action Spaces: Optimality Conditions Based on Multimodularity," SIAM J. on Control and Optimization, Vol. 38, pp. 875-892.

[Wat89] Watkins, C. J. C. H., Learning from Delayed Rewards, Ph.D. Thesis, Cambridge Univ., England.

[WeB99] Weaver, L., and Baxter, J., 1999. "Learning from State Differences: STD(λ)," Tech. Report, Dept. of Computer Science, Australian National University.

[WhS94] White, C. C., and Scherer, W. T., 1994. "Finite-Memory Suboptimal Design for Partially Observed Markov Decision Processes," Operations Research, Vol. 42, pp. 439-455.

[Whi88] Whittle, P., 1988. "Restless Bandits: Activity Allocation in a Changing World," J. of Applied Probability, pp. 287-298.

[Whi91] White, C. C., 1991. "A Survey of Solution Techniques for the Partially Observed Markov Decision Process," Annals of Operations Research, Vol. 32, pp. 215-230.

[WiB93] Williams, R. J., and Baird, L. C., 1993. "Analysis of Some Incremental Variants of Policy Iteration: First Steps Toward Understanding Actor-Critic Learning Systems," Report NU-CCS-93-11, College of Computer Science, Northeastern University, Boston, MA.

[Wie03] Wiewiora, E., 2003. "Potential-Based Shaping and Q-Value Initialization are Equivalent," J. of Artificial Intelligence Research, Vol. 19, pp. 205-208.

[Wit66] Witsenhausen, H. S., 1966. Minimax Control of Uncertain Systems, Ph.D. thesis, MIT.

[Wit68] Witsenhausen, H., 1968. "A Counterexample in Stochastic Optimum Control," SIAM 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.

[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.

[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.

[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.

[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 preprint arXiv:1911.01004.

[Yu14] Yu, H., 2014. "Stochastic Shortest Path Games and Q-Learning," arXiv preprint 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 preprint arXiv: 1611.03530.

[ZBH21] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O., 2021. "Understanding Deep Learning (Still) Requires Rethinking Generalization," Communications of the ACM, VOL. 64, pp. 107-115.

[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.

[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.