July 2021

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

Dimitri Bertsekas ††

Abstract

Some of the most exciting success stories in reinforcement learning have been in the area of games. Primary examples are the recent AlphaZero program (which plays chess), and the similarly structured and earlier (1990s) TD-Gammon program (which plays backgammon). These programs were trained off-line extensively using sophisticated approximate policy iteration algorithms and neural networks. Yet the AlphaZero player that has been obtained off-line is not used directly during on-line play. Instead a separate on-line player is used, which is based on multistep lookahead and a terminal cost that was trained using experience with the off-line player. The on-line player has greatly improved performance. Similarly, TD-Gammon computed off-line a terminal cost function approximation, which was used to extend its on-line lookahead by rollout (simulation with the one-step lookahead player that is based on the terminal cost function approximation). In particular:

(a) The on-line player of AlphaZero plays much better than its extensively trained off-line player. This is due to the beneficial effect of approximation in value space with long lookahead minimization, which corrects for the inevitable imperfections of the off-line player, and its terminal cost approximation.

(b) The TD-Gammon player that uses long rollout plays much better than TD-Gammon with one-step or two-step lookahead 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 performance of an off-line trained controller can be greatly improved by on-line approximation in value space, with long lookahead (whether involving minimization or rollout with an off-line obtained policy), and terminal cost approximation that is obtained off-line. This performance enhancement is often dramatic and is due to a simple fact, which is the focal point of this paper: *approximation in value space 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 and on-line rollout*. This process can be understood in terms of abstract models of infinite horizon dynamic programming and simple geometrical constructions. It manifests itself to some extent in model predictive control, but it seems that it has yet to be fully appreciated within

† Early draft of a research monograph to be published by Athena Scientific, Belmont, MA, sometime in 2022. The complete monograph will include expanded versions of Sections 4 and 5, and a treatment of finite horizon and discrete optimization problems.

†† Fulton Professor of Computational Decision Making, ASU, Tempe, AZ, and McAfee Professor of Engineering, MIT, Cambridge, MA.

the decision and control community.

There is an additional benefit of policy improvement by approximation in value space, not observed in the context of games (which have stable rules and environment). It works well with changing problem parameters and on-line replanning, similar to indirect adaptive control. Here the Bellman equation is perturbed due to the parameter changes, but approximation in value space still operates as a Newton step.

In this paper we aim to provide analysis and insights (often based on visualization), which explain the beneficial effects of on-line decision making on top of off-line training. In particular, through a unifying abstract mathematical framework, we show that the principal AlphaZero/TD-Gammon ideas of approximation in value space and rollout apply very broadly to deterministic and stochastic optimal control problems, involving both discrete and continuous search spaces. Moreover, these ideas can be effectively integrated with other important methodologies such as model predictive control, adaptive control, decentralized control, discrete and Bayesian optimization, neural network-based value and policy approximations, and heuristic algorithms for discrete optimization.

Contents

1. ALPHAZERO, OFF-LINE TRAINING, AND ON-LINE PLAY

Perhaps the most impressive success story in reinforcement learning (RL) is the development of the AlphaZero program by DeepMind Inc; see [SHS17], [SSS17]. AlphaZero plays Chess, Go, and other games, and is an improvement in terms of performance and generality over the earlier AlphaGo program [SHM16], which plays the game of Go only. AlphaZero plays 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).

We should note also that the principles of the AlphaZero design have much in common with the TD-Gammon program of Tesauro [Tes94], [Tes95], [TeG96] that plays backgammon (a game of substantial computational and strategical complexity, which involves a number of states estimated to be in excess of 10^{20}). 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 program for the (one-player) game of Tetris, based on similar principles, is described by Scherrer et al. [SGG15], together with several antecedents. The backgammon and tetris programs, while dealing with less complex games than chess, are of special interest because they involve significant stochastic uncertainty, and are thus unsuitable for the use of long lookahead, which is widely believed to be one of the major contributors to the success of AlphaZero, and chess programs in general.

Still, for all of their impressive achievements and brilliant implementations, these game programs are couched on well established methodology, from optimal and suboptimal control, which is portable to far broader domains of engineering, economics, and other fields. This is the methodology of dynamic programming (DP), policy iteration, limited lookahead, rollout, and related approximations in value space. The aim of this paper is to propose a conceptual, somewhat abstract framework, which allows insight into the connections of AlphaZero and TD-Gammon with some of the core problems in decision and control.

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

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

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

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

AlphaZero for the most part.

1.1 Off-Line Training and Policy Iteration

An off-line training algorithm like the one used in AlphaZero is the part of the program that learns how to play through self-training that takes place before real-time play against any opponent. It is illustrated in Fig. 1.1, and it generates a sequence of *chess players* and *position evaluators*. A chess player assigns "probabilities" to all possible moves at any given chess position (these are the probabilities with which the player selects the possible moves at the given position). A position evaluator assigns a numerical score to any given chess position, and thus predicts quantitatively the performance of a player starting from any position. The chess player and the position evaluator are represented by neural networks, a *policy network* and a *value network*, which accept a chess position and generate a set of move probabilities and a position evaluation, respectively.†

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

The overall training algorithm is a form of *policy iteration*, a DP algorithm that will be of primary interest to us in this paper. Starting from a given player, it repeatedly generates (approximately) improved players, and settles on a final player that is judged empirically to be "best" out of all the players generated. Policy iteration may be separated conceptually into two stages (see Fig. 1.1).

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

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

‡ One more complication is that chess and Go are two-player games, while most of our development will involve single-player optimization. However, DP theory and algorithms extend to two-player games, although we will not discuss these extensions.

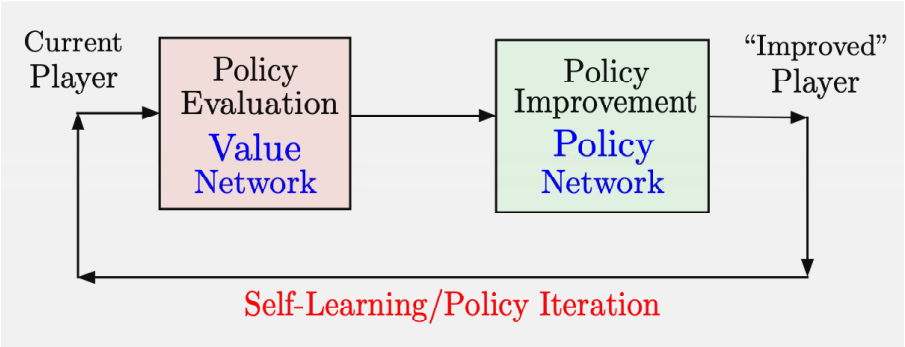

Figure 1.1 Illustration of the AlphaZero off-line training algorithm. It generates a sequence of position evaluators and chess players. The position evaluator and the chess player are represented by two neural networks, a value network and a policy network, which accept a chess position and generate a position evaluation and a set of move probabilities, respectively.

(b) *Policy improvement*: Given the current player and its position evaluator, trial move sequences are selected and evaluated for the remainder of the game starting from many positions. An improved player is then generated by adjusting the move probabilities of the current player towards the trial moves that have yielded the best results.

In AlphaZero the policy evaluation is done by using deep neural networks. The policy improvement uses a complicated algorithm called *Monte Carlo Tree Search* (MCTS for short), a form of randomized multistep lookahead that enhances the efficiency of the multistep lookahead operation, by pruning intelligently the multistep lookahead tree. However, deep neural networks and MCTS, which leading to some performance gains, are not of fundamental importance. The approximation quality that a deep neural network can achieve can also be achieved with a shallow neural network, perhaps with reduced sample efficiency. Similarly MTCS cannot achieve better lookahead accuracy than standard exhaustive search, although it may be more efficient computationally. Indeed, policy improvement can be done more simply without MCTS, as in Tesauro's TD-Gammon program: we 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 current 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.†

† 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

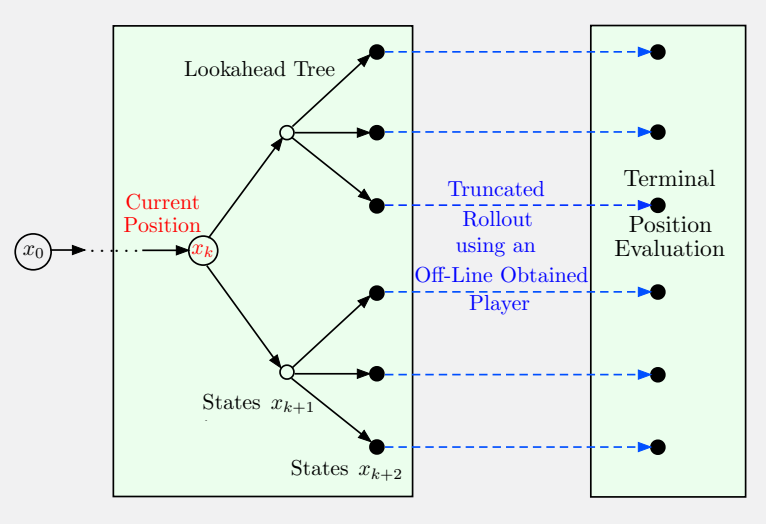

Figure 1.2 Illustration of an on-line player such as the one used in AlphaGo, AlphaZero, and Tesauro's backgammon program [TeG96]. At a given position, it generates a lookahead tree of multiple moves up to a given depth, then runs the off-line obtained player for some more moves, and then evaluates the effect of the remaining moves by using the position evaluator of the off-line obtained player.

1.2 On-Line Play and Approximation in Value Space - Truncated Rollout

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

Given the policy network/player obtained off-line and its value network/position evaluator, the on-line algorithm plays roughly as follows (see Fig. 1.2). 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

parameters to match the search probabilities and selfplay game outcome respectively."

† Quoting from the paper [SSS17]: "The MCTS search outputs probabilities of playing each move. These search probabilities usually select much stronger moves than the raw move probabilities of the neural network." To elaborate, this statement refers to the MCTS algorithm that is used on line to generate the move probabilities at each position encountered in the course of a given game. The neural network referred to is trained off-line, also using in part the MCTS algorithm.

position evaluator of the value network. The middle portion, called "truncated rollout," may be viewed as *an economical substitute for longer lookahead*. Actually truncated rollout is not used in the published version of AlphaZero [SHS17]; the first portion (multistep lookahead) is very long and implemented efficiently (though the use of MCTS), so that the rollout portion is not essential. However, rollout is used in AlphaGo [SHM16]. Moreover, chess and Go programs (including AlphaZero) typically use a well-known limited form of rollout, called "quiescence search," which aims to resolve imminent threats and highly dynamic positions through simulated multi-move piece exchanges, before invoking the position evaluator. Rollout is instrumental in achieving high performance in Tesauro's 1996 backgammon program [TeG96]. The reason is that backgammon involves stochastic uncertainty, so long lookahead is not possible because of rapid expansion of the lookahead tree with every move.†

The DP ideas with cost function approximations, similar to the on-line player illustrated in Fig. 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 this paper.‡

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

1.3 The Lessons of AlphaZero

The AlphaZero and TD-Gammon experiences reinforce an important conclusion that applies more generally to decision and control problems: despite the extensive off-line effort that may have gone into the design of a

† Tesauro's rollout-based backgammon program [TeG96] uses only a value network, called TD-Gammon, which was trained using an approximate policy iteration scheme developed several years earlier [Tes94]. TD-Gammon is used to generate moves for the truncated rollout via a one-step or two-step lookahead minimization. Thus the value network also serves as a substitute for the policy network during the rollout operation. The terminal position evaluation used at the end of the truncated rollout is also provided by the value network. The middle portion of Tesauro's scheme (truncated rollout) is important for achieving a very high quality of play, as it effectively extends the length of lookahead from the current position.

‡ 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. However, as a methodological subject, approximation in policy space has little connection to the ideas of the present paper.

policy, its performance may be greatly improved by on-line approximation in value space, with either extra lookahead involving minimization and/or with rollout using this policy, and terminal cost approximation.

In the following sections, we will aim to amplify on this theme and to focus on the principal characteristics of AlphaZero-like architectures, within a broader context of optimal decision and control. We will make use of intuitive visualization, and the central role of Newton's method for solving Bellman's equation.† We will aim to show that feedback control, based on approximation in value space and the underlying off-line training/on-line play structure, offers benefits that go well beyond the conventional control theory wisdom that "feedback corrects for noise, uncertainty, and modeling errors."

Our mathematical framework is couched on abstract DP, including abstract forms of Bellman's equation, and the value and policy iteration algorithms (see the author's books [Ber12], [Ber18a]). However, in this paper, we will deemphasize mathematical proofs and derivations. There is considerable related analysis, which supports our conclusions and can be found in the author's recent RL books [Ber19a], [Ber20a].

In summary, our discussion will aim to highlight the following points:

Summary

(a) Approximation in value space is a single step of Newton's method for solving Bellman's equation (perhaps preceded by preliminary adjustments and/or extra steps of lookahead to enhance the starting point of the Newton step).

(b) The starting point for the Newton step of (a) is obtained by some unspecified methodology, which may involve the solution of a related but simpler problem, and/or training with data that makes use of neural networks or feature-based architectures.

(c) The on-line play and off-line training parts of the AlphaZero/TD-Gammon design structure correspond to (a) and (b) above, respectively.

(d) The on-line player of AphaZero plays much better than its deep neural network-trained player for the same reason that the Newton step (a) improves substantially on its starting point (b), namely the underlying superlinear convergence property that is typical of Newton's method.

(e) Approximation in value space with ℓ-step lookahead can be viewed as one-step lookahead where $\ell - 1$ value iterations are used to enhance the starting point of the Newton step of (a) above.

† Bellman's equation, the centerpiece of infinite horizon DP theory, is viewed here as a functional equation, whose solution is the cost of operating the system viewed as a function of the system's initial state. We will give examples of Bellman's equation in Section 2 for discounted and other problems, and we will also provide in Section 3 abstract forms of Bellman's equation that apply more generally.

(f) The algorithmic processes for (a) and (b) above can be designed by a variety of methods, and independently of each other. For example:

(1) The implementation of the Newton step (a) may or may not involve any of the following: truncated rollout, on-line Monte Carlo simulation, MCTS or other efficient tree search techniques, on-line policy iteration, etc.

(2) The computation of the starting point (b) may or may not involve any of the following: Q-learning, approximate policy iteration based on temporal differences or aggregation, neural networks, feature-based function approximation, policies trained off-line by approximation in policy space, including policy gradient methods or policy random search, etc. Moreover, the details of this computation may vary broadly without affecting significantly the effectiveness of the overall scheme, which is primarily determined by the Newton step (a).

(g) An efficient implementation of the Newton step (a) is often critical in order to meet real-time constraints for generating controls, and to allow longer lookahead minimization, which typically enhances the starting point of the Newton step and its performance. By contrast, off-line training algorithms used for (b) have much less stringent real-time constraints, and the issues of sample efficiency and fine tuned performance, while important, are not critical.

(h) Approximation in value space addresses effectively issues of robustness and on-line replanning for problems with changing parameters. The mechanism is similar to the one of indirect adaptive control: changing problem parameters are estimated on-line and a Newton step is used in place of an expensive full reoptimization of the controller. In the presence of changing parameters, the starting point for the Newton step changes, but the Newton step itself remains powerful and aims at the optimal solution that corresponds to the estimated system parameters.

(i) Model predictive control (MPC) has a conceptually similar structure to the AlphaZero-like programs, and entails an on-line play component involving multistep minimization, and an off-line training component to construct terminal cost approximations, and "safe" state space regions or tubes. The success of MPC may be attributed to these similarities and to its resilience to changing problem parameters as per (h) above.

(j) Because the ideas outlined above are couched on principles of DP that often hold for arbitrary state and control spaces, they are valid within very general contexts: continuous-spaces control systems, discrete-spaces Markovian decision problems, control of hybrid systems, and discrete and combinatorial optimization.

The preceding points are meant to highlight the essence of the connections between AlphaZero and TD-Gammon, approximation in value space, and decision and control. Naturally in practice there are exceptions and modifications, which need to be worked out in the context of particular applications, under appropriate assumptions. Moreover, while some results and elaborations are available through the work that has been done on approximate DP and on MPC, several of the results suggested by the analysis and insights of the present paper remain to be rigorously established and enhanced within the context of specific problems.

The paper is structured as follows. In Section 2 we review the theory of classical infinite horizon optimal control problems, in order to provide some orientation and an analytical platform for what follows in subsequent sections. In Section 3, we introduce an abstract DP framework that will set the stage for the conceptual and visual interpretations of approximation in value space in terms of Newton's method. In Section 4, we discuss various issues of changing problem parameters, adaptive control, and MPC. Finally, in Section 5, we discuss on-line policy iteration, which aims to improve the on-line approximation in value space algorithm by using training data that is collected on-line.

2. DETERMINISTIC AND STOCHASTIC DYNAMIC PROGRAMMING

In this section we will describe a classical optimal control framework, illustrated in Fig. 2.1, which we will use as a principal example for a more abstract DP framework to be introduced in Section 3. This abstract framework will be used in turn as the starting point for our analysis and visualization of algorithmic issues, relating to approximation in value space, multistep lookahead, controller stability, truncated rollout, and policy iteration.

2.1 Optimal Control Over an Infinite Horizon

Let us consider a familiar class of stochastic optimal control problems over an infinite horizon. We have a stationary system of the form

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

where x_k is an element of some state space X and the control u_k is an element of some control space U. The system includes a random "disturbance" w_k with a probability distribution $P(\cdot \mid x_k, u_k)$ that may depend explicitly on x_k and u_k, but not on values of prior disturbances $w_{k-1}, \ldots, w_0$. The control u_k is constrained to take values in a given subset $U(x_k) \subset U$, which depends on the current state x_k. We are interested in policies $\pi = \{\mu_0, \mu_1, \ldots\}$, such that each function μ_k maps states into controls, and satisfies $\mu_k(x_k) \in U(x_k)$ for all k. A stationary policy of the form $\{\mu, \mu, \ldots\}$ will also be referred to as "policy μ." We make no assumptions on the state, control, and disturbances, and indeed for most of the discussion of this paper, these spaces can be arbitrary.

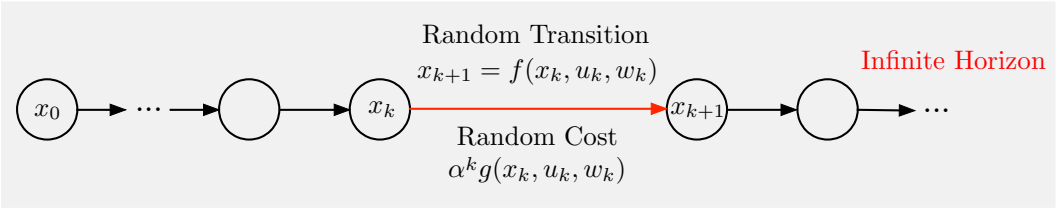

Figure 2.1 Illustration of an infinite horizon problem. The system and cost per stage are stationary, except for the use of a discount factor α. If $\alpha = 1$, there is typically a special cost-free termination state that we aim to reach.

We aim to minimize the expected total cost over an infinite number of stages, given by

$$J_\pi(x_0) = \lim_{N\to\infty} E\left\{\sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k), w_k\big)\right\}, \tag{2.1}$$

where $\alpha^k g(x_k, u_k, w_k)$ is the cost of stage k, and $\alpha \in (0, 1]$ is a discount factor. If $\alpha = 1$ we refer to the problem as undiscounted. The expected value in Eq. (2.1) is taken with respect to the random disturbances w_k, $k = 0, 1, \ldots$. Here, $J_\pi(x_0)$ denotes the cost associated with an initial state x_0 and a policy $\pi = \{\mu_0, \mu_1, \ldots\}$. The cost function of a stationary policy μ is denoted by J_μ. The optimal cost starting at state x, $\inf_\pi J_\pi(x)$, is denoted by $J^*(x)$, and the function J^* is referred to as the *optimal cost function*.

Let us consider some special cases:

(a) *Stochastic shortest path problems* (SSP for short). Here, $\alpha = 1$ but there is a special cost-free termination state, denoted by t; once the system reaches t it remains there at no further cost. Usually, the termination state t represents a goal state that we are trying to reach at minimum cost; these are problems where the cost per stage is nonnegative, and will of primary interest in this paper. In some other types of problems, t may be a state that we are trying to avoid for as long as possible; these are problems where the cost per stage is nonpositive, and will not be specifically discussed in this paper.

(b) *Discounted stochastic problems*. Here, $\alpha < 1$ and there need not be a termination state. However, there is a substantial connection between SSP and discounted problems. Aside from the fact that they are both infinite horizon total cost optimization problems, 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. Thus SSP and discounted problems share qualitative similarities in their respective theories.

(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

$$x_{k+1} = f(x_k, u_k), \qquad k = 0, 1, \ldots, \tag{2.2}$$

and

$$J_\pi(x_0) = \lim_{N\to\infty} \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k)\big). \tag{2.3}$$

We assume further that there is a termination state t, and we have

$$g(x, u) \geq 0, \qquad \text{for all } x \neq t,\ u \in U(x), \tag{2.4}$$

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]. The best known special case is the classical linear quadratic problem where the termination state is $t = 0$, the system is linear of the form

$$x_{k+1} = Ax_k + Bu_k, \tag{2.5}$$

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, B is an $n \times m$ matrix, and the cost per stage is quadratic of the form

$$g(x.u) = x'Qx + u'Ru, \tag{2.6}$$

where Q and R are positive semidefinite symmetric matrices of dimension $n\times n$ and $m\times m$, respectively (all finite-dimensional vectors in this paper are viewed as column vectors, and a prime denotes transposition). Another important problem, finding a shortest path from an origin node to a destination node in a graph, is obtained when the number of states is finite.

Infinite Horizon Methodology

Many of the analytical and computational issues regarding infinite horizon problems revolve around the relation between the optimal cost function J^* of the problem and the optimal cost function of the corresponding N-stage problem. 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 *value iteration* algorithm (VI for short)

$$J_{k+1}(x) = \min_{u\in U(x)} E\Big\{g(x, u, w) + \alpha J_k\big(f(x, u, w)\big)\Big\}, \qquad k = 0, 1, \ldots, \tag{2.7}$$

starting from $J_0(x) \equiv 0$. 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.,

$$J^*(x) = \lim_{N\to\infty} J_N(x) \tag{2.8}$$

for all states x.

(2) Bellman's equation holds for all states x:

$$J^*(x) = \min_{u \in U(x)} E\Big\{g(x,u,w) + \alpha J^*\big(f(x,u,w)\big)\Big\}. \tag{2.9}$$

(3) If $\mu(x)$ attains the minimum in the right-hand side of the Bellman equation (2.9) for each x, then the stationary policy μ should be optimal.

All three of the preceding results hold for discounted problems, provided the expected cost per stage $E\big\{g(x,u,w)\big\}$ is bounded over the set of possible values of (x,u,w) (see [Ber12], Chapter 1). They also hold for finite-state SSP problems under reasonable assumptions. For deterministic problems with possibly infinite state and control spaces, there is substantial analysis that provides assumptions under which the results (1)-(3) above hold. The author's paper [Ber17b] considers the undiscounted special case of the problem (2.2)-(2.4), where there is a cost-free and absorbing termination state t, the cost function is strictly positive for all other states, as in Eq. (2.4), and the objective is to either reach or asymptotically approach t.

The VI algorithm is also typically valid, in the sense that $J_k \to J^*$, even if the initial function J_0 is nonzero. The motivation for a different choice of J_0 is faster convergence to J^*; generally the convergence is faster as J_0 is chosen closer to J^*. The intuitive interpretation of the Bellman equation (2.9) is that it is the limit as $k \to \infty$ of the VI algorithm (2.7) assuming that $J_k \to J^*$. The optimality condition (3) indicates that optimal and near optimal policies can be obtained from within the class of stationary policies, something that is generally true for the problems that we discuss in this paper, and that we will implicitly assume in what follows.

Aside from the VI algorithm, another fundamental algorithm is policy iteration (PI for short), which will be discussed in Section 3.3. It can be viewed as Newton's method for solving Bellman's equation, and is central for our purposes.

2.2 Approximation in Value Space - One-Step and Multistep Lookahead - Stability

A principal RL approach to deal with the often intractable exact computation of J^* is *approximation in value space*. 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*

$$\tilde{\mu}(x) \in \arg\min_{u \in U(x)} E\Big\{g(x,u,w) + \alpha \tilde{J}\big(f(x,u,w)\big)\Big\}. \tag{2.10}$$

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 cost obtained when using $\tilde{\mu}$ starting at state x]. In the next section, the change from $\tilde{J}$ to $J_{\tilde{\mu}}$ will be interpreted

as a step of Newton's method for solving Bellman's equation. Among others, this will suggest that $J_{\tilde{\mu}}$ is close to J^* and obeys a superlinear convergence relation

$$\lim_{\tilde{J}\to J^*} \frac{J_{\tilde{\mu}}(x) - J^*(x)}{\tilde{J}(x) - J^*(x)} = 0,$$

for all states x. For specific types of problems, this relation represents a plausible result, which likely holds under appropriate conditions. This is similar to the use of Newton's method in numerical analysis, where its global or local convergence is guaranteed only under some assumptions. Within our context of approximate DP, however, there is an important underlying structure, which enhances the convergence properties of Newton's method, namely the monotonicity and concavity properties of Bellman's equation, as we will discuss in what follows.

While it is desirable that $J_{\tilde{\mu}}$ is close to J^* in some sense, for classical control problems involving control to a desirable 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. For the purposes of this paper, we will focus on stability issues for just this one class of problems, and *we will consider the policy $\tilde{\mu}$ to be stable if*

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

Selecting $\tilde{J}$ so that $\tilde{\mu}$ is stable is a question of major interest, and will be addressed in Section 3.

ℓ-Step Lookahead

An important extension of one-step lookahead 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. 2.2). Actually, we 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 $\tilde{J}(x_{k+\ell})$ on the state $x_{k+\ell}$ obtained after $\ell - 1$ stages. In the next section, this will be interpreted as a step of Newton's method for solving Bellman's equation, starting from a function $\hat{J}$, which is an "improvement" over $\tilde{J}$. In particular, $\hat{J}$ is obtained from $\tilde{J}$ by applying $\ell - 1$ successive value iterations.

The motivation for ℓ-step lookahead 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 in the next section, 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. However, the solution of the multistep lookahead optimization problem, instead of the one-step lookahead counterpart of Eq. (2.10), becomes more time consuming.

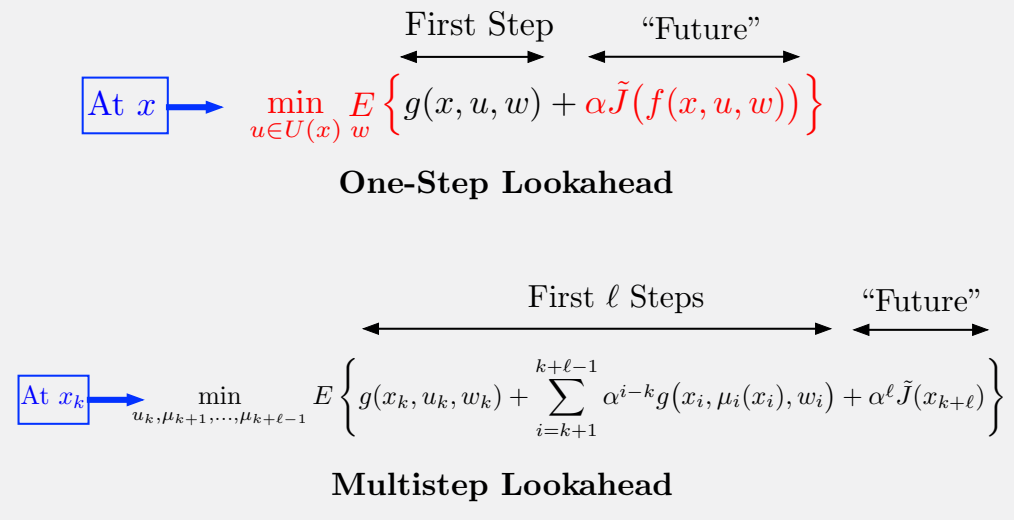

Figure 2.2 Schematic illustration of approximation in value space with one-step and ℓ-step lookahead.

Constructing Terminal Cost Approximations

A major issue in value space approximation is the construction of suitable approximate cost-to-go functions $\tilde{J}$. 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, 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 terminal cost approximation. While the method by which we obtain $\tilde{J}$ will not be important for understanding the ideas of this paper, for orientation purposes we briefly describe three broad types of approximation, and refer to the RL and approximate DP literature for further details:

(a) *Problem approximation*: Here the function $\tilde{J}$ is obtained 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, and ignoring various types of uncertainties; for example we may consider using as $\tilde{J}$ the cost function of a related deterministic problem, 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 to the optimal cost function of the original problem, using some form of interpolation.

(b) *On-line approximate optimization*, such as rollout algorithms and model predictive control, which will be discussed in more detail later. These methods often involve the use of a suboptimal policy μ, which is applied on-line when needed to compute the values $\tilde{J}(x)$ to be exactly or approximately equal to $J_\mu(x)$. The policy μ may be obtained by any method, e.g., one based on heuristic reasoning, or on a more principled approach, such as approximation in policy space.

(c) *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.

We refer to the neurodynamic programming book by Bertsekas and Tsitsiklis [BeT96], and the RL book by Sutton and Barto [SuB18], as well as the large number of subsequent RL and approximate DP books, which provide specific examples of cost function approximation methods and associated training algorithms.

From Off-Line Training to On-Line Play

Generally off-line training will produce either just a policy, or just a cost approximation (as in the case of TD-Gammon), or both (as in the case of AlphaZero). We have already discussed in this section one-step lookahead and multistep lookahead schemes to implement on-line approximation in value space using $\tilde{J}$; cf. Fig. 2.2. Let us now consider some additional possibilities, some of which involve the use of a policy μ that has been obtained off-line (possibly in addition to a terminal cost approximation). Here are some of the main possibilities:

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

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

that are needed for the case of one-step lookahead [cf. Eq. (2.10)], or the values of

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

that are needed for the case of ℓ-step lookahead. This is the simplest form of rollout, and only requires the off-line construction of the policy μ.

(b) *Given a terminal cost approximation* $\tilde{J}$ that has been obtained off-line, we may use it on-line to compute a one-step or multistep lookahead policy $\tilde{\mu}$. This policy can in turn be used for rollout as in (a) above. In a variation of this scheme, we may also use $\tilde{J}$ for truncated rollout, to approximate the tail end of the rollout process (an example of this is the rollout-based TD-Gammon algorithm discussed in Section 1.2).

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

The preceding three possibilities are the principal ones for using the results of off-line training in on-line play schemes. Naturally, there are variations where additional information is computed off-line to facilitate and/or expedite the on-line play algorithm. As an example, in MPC, in addition to a terminal cost approximation, a target tube may need to be computed off-line in order to guarantee that some state constraints can be satisfied on-line; see the discussion of Section 4.2. Other examples of this type will be noted in the context of specific applications.

3. AN ABSTRACT VIEW OF REINFORCEMENT LEARNING

In this section 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 multi-step lookahead policy $\tilde{\mu}$. To understand these constructions, we need an abstract notational framework. In particular, we denote by TJ the function in the right-hand side of Bellman's equation. Its value at state x is given by

$$(TJ)(x) = \min_{u \in U(x)} E\Big\{ g(x,u,w) + \alpha J\big(f(x,u,w)\big) \Big\}; \tag{3.1}$$

(throughout the paper, we will be using "min" instead of the more formal "inf," even we are not sure that minimum is attained). Also for each policy μ, we introduce the corresponding function $T_\mu J$, which has value at x given by

$$(T_\mu J)(x) = E\Big\{ g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big) \Big\}. \tag{3.2}$$

Thus T and T_μ can be viewed as operators (referred to as the *Bellman operators*), which map functions J to other functions (TJ or $T_\mu J$, respectively). These operators are illustrated in Fig. 3.1.†

† Within the context of this paper, 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. (3.1) and (3.2) are well-defined and finite when J is

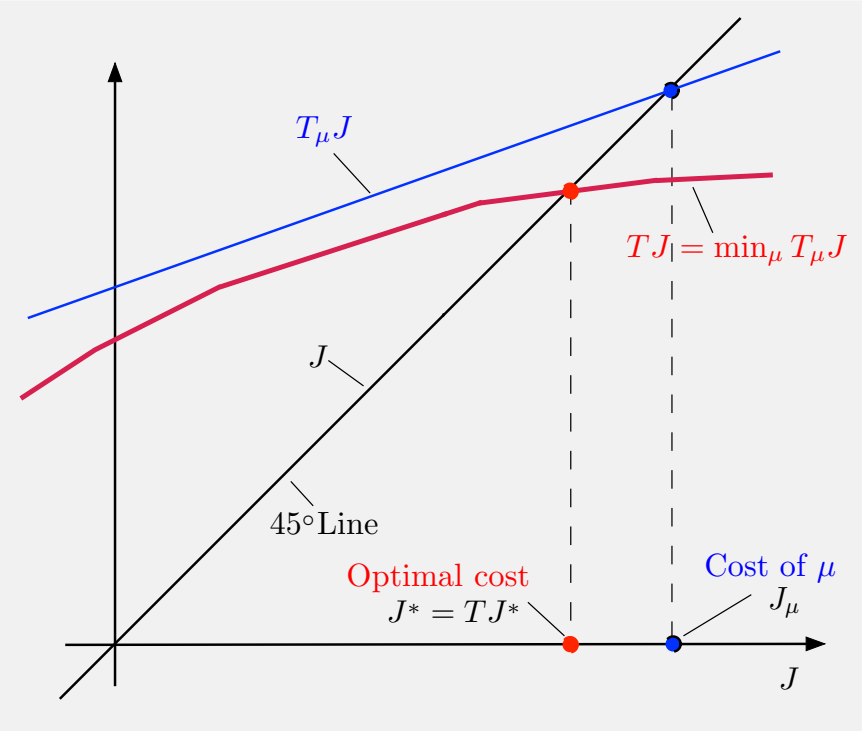

Figure 3.1 Geometric interpretation of the Bellman operators T_μ and T, the Bellman equations, and the VI algorithm. The functions J, J^*, TJ, $T_\mu J$, etc, are multidimensional (they have as many scalar components as there are states), but they are shown projected onto one dimension. The function $T_\mu J$ for each policy μ is linear (assuming it is real-valued), while the function TJ can be written as $\min_\mu T_\mu J$ and its components, $(TJ)(x)$, are concave as functions of J for every x. 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. Similarly, J_μ is obtained from the intersection of the graph of $T_\mu J$ and the 45 degree line. Clearly, we have $J^* \leq J_\mu$ for all μ.

The figure above 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.

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

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

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. (3.1). We allow this possibility, although our illustrations will primarily depict the case where TJ is real-valued. Generally, the theory of abstract DP is developed with the use of extended real-valued functions; see [Ber18a].

then we have

$$(TJ)(x) \geq (TJ')(x), \qquad (T_\mu J)(x) \geq (T_\mu J')(x), \qquad \text{for all } x \text{ and } \mu. \tag{3.3}$$

Moreover, an important point is that *the Bellman equation $J = T_\mu J$ is linear in J*, while the function TJ can be written as $\min_\mu T_\mu J$ and its components, $\min_\mu (T_\mu J)(x)$, are concave for all x.†

For 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, using transition probability notation, whereby $p_{ij}(u)$ denotes the transition probability from state i to state j under control u, the operator T_μ takes the form

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

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

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

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

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

Clearly, $(TJ)(1)$ and $(TJ)(2)$ are concave and piecewise linear 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 because $(TJ)(x)$ is the minimum of a collection of linear functions of J, one for each control $u \in U(x)$.

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

$$\big\{(J,\xi) \in R(X) \times R(X) \mid (TJ)(x) \geq \xi(x), \text{ for all } x \in X\big\} \tag{3.4}$$

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. 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 for example in discounted problems with bounded cost per

† 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.

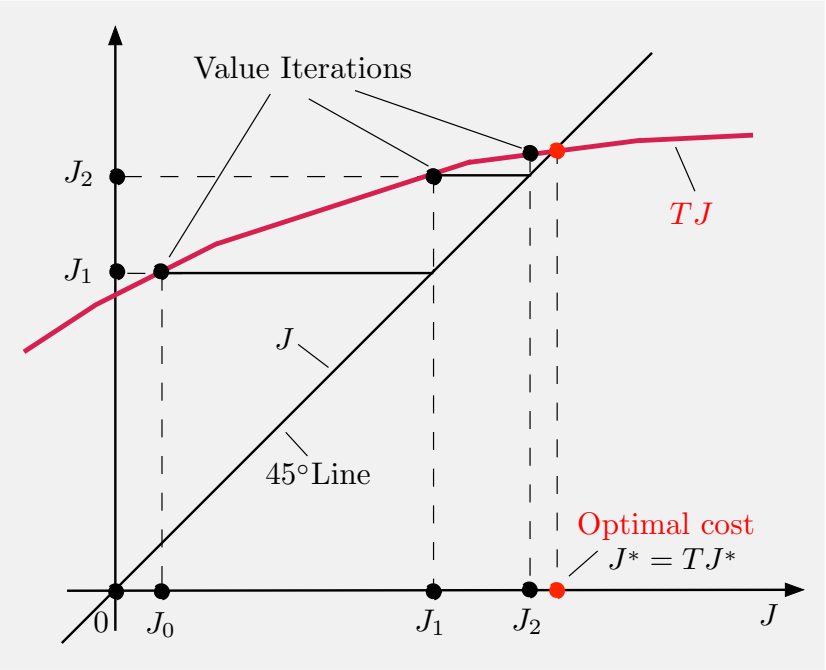

Figure 3.2 Geometric interpretation of the value iteration algorithm $J_{k+1} = TJ_k$, starting from some initial function J_0. Successive iterates are obtained through the staircase construction shown in the figure. The value iteration algorithm $J_{k+1} = T_\mu J_k$ for a given policy μ can be similarly interpreted, except that the graph of the function $T_\mu J$ is linear.

stage. However, for undiscounted problems, asserting the contraction property of T or T_μ may be more complicated.

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

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

cf. the geometric interpretation of Fig. 3.2. Moreover, the VI algorithm for a given policy 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. To keep the presentation simple, we will focus our attention on the abstract DP framework as it applies to the optimal control problems of Section 2.1. In particular, we will assume without further mention that T *and* T_μ *have the monotonicity property (3.3), 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 paper. Such models include

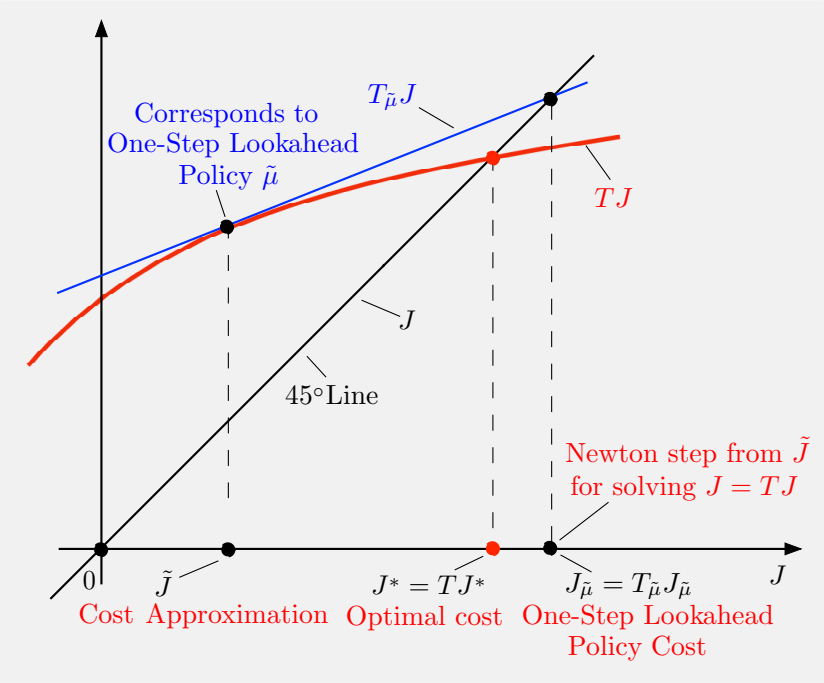

Figure 3.3 Geometric interpretation of approximation in value space and the one-step lookahead policy $\tilde{\mu}$ as a step of Newton's method [cf. Eq. (2.10)]. Giver $\tilde{J}$, we find the policy $\tilde{\mu}$ that attains the minimum in the relation $T\tilde{J} = \min_\mu T_\mu \tilde{J}$. This policy is the one for which $T\tilde{J} = T_{\tilde{\mu}}\tilde{J}$, so the graph of TJ and $T_{\tilde{\mu}}J$ touch at $\tilde{J}$, as shown. 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 $J_{\tilde{\mu}}$.

semi-Markov problems, minimax control problems, risk sensitive problems, Markov games, and others (see the DP textbook [Ber12], and the abstract DP monograph [Ber18a]).

3.1 Approximation in Value Space and Newton's Method

Let us now interpret approximation in value space in terms of abstract geometric constructions. In particular, by using the operators T and T_μ, the one-step lookahead policy $\tilde{\mu}$ is characterized by the equation

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

as in Fig. 3.3 [cf. Eq. (2.10)]. Thus the relation $T\tilde{J} = T_{\tilde{\mu}}\tilde{J}$ implies that the graph of $T_{\tilde{\mu}}J$ just touches the graph of TJ at $\tilde{J}$, as shown in the figure [mathematically, the graph of $T_{\tilde{\mu}}J$ defines a hyperplane within the space of function pairs $R(X) \times R(X)$ that supports the convex set (3.4) at the point $\big(\tilde{J}, T(\tilde{J})\big)$]. Since $T_{\tilde{\mu}}J$ is a linear function, the equation $J = T_{\tilde{\mu}}J$ is a linearization of the equation $J = TJ$ at the point $\tilde{J}$. Its solution $J_{\tilde{\mu}}$, can be viewed as the result of a Newton iteration at the point $\tilde{J}$: 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-

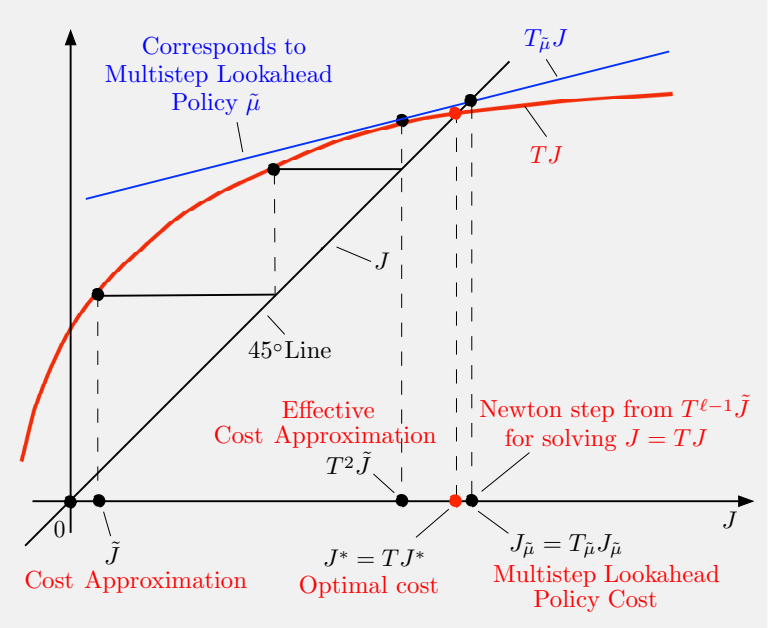

Figure 3.4 Geometric interpretation of approximation in value space with ℓ-step lookahead (in this figure $\ell = 3$). It is the same as approximation in value space with one-step lookahead using $T^{\ell-1}\tilde{J}$ as cost approximation. It can be viewed as a Newton step at the point $T^{\ell-1}\tilde{J}$, 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 $\lim_{\ell\to\infty} J_{\tilde{\mu}} = J^*$.

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.

The structure of the Bellman operators (3.1) and (3.2), 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].

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. 3.4.†

3.2 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 paper, 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. (2.2)-(2.4)]. 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, the policy μ is called *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 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 approximations $\tilde{J}$ for which the one-step or multistep lookahead policy $\tilde{\mu}$ is 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 in Section 3.4 for the undiscounted deterministic case of the linear quadratic problem of Section 2.1 [cf. Eqs. (2.5), (2.6)].

Figure 3.5 illustrates the region of stability for approximation in value space with one-step lookahead. An important fact is that assuming that the cost improvement inequality

$$J^* \leq J_{\tilde{\mu}} \leq J_\mu,$$

† Variants of Newton's method that involve multiple fixed point iterations, before and after each Newton step are also well-known in numerical analysis. They are viewed as combinations of first order iterative methods, such as the Gauss-Seidel and Jacobi algorithms, and Newton's method, and they are called Newton-SOR methods in the classic book by Ortega and Rheinboldt [OrR70] (Section 13.4).

‡ For the undiscounted positive cost deterministic problem introduced earlier [cf. Eqs. (2.2)-(2.4)], 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 Bertsekas [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.

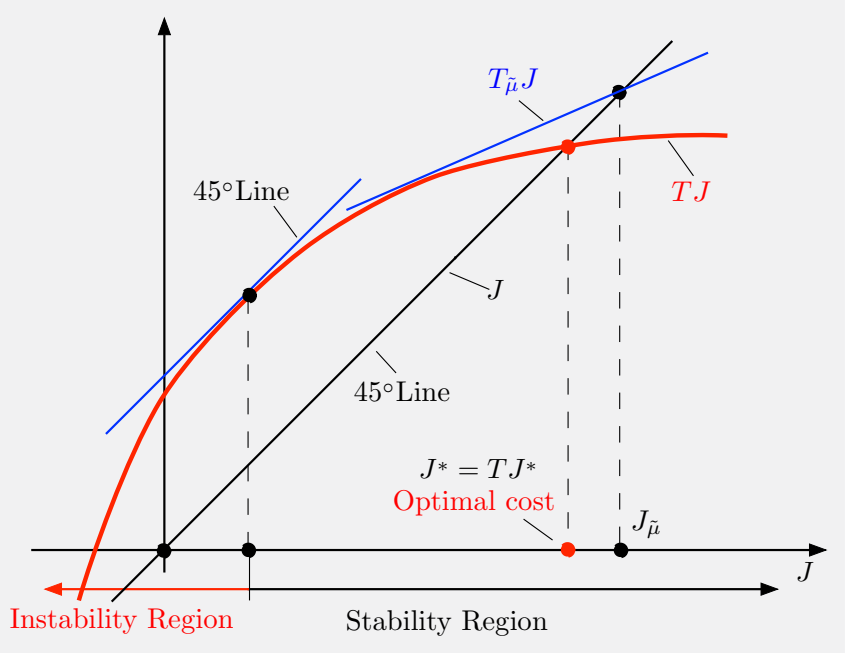

Figure 3.5 Illustration of the regions of stability and instability for approximation in value space with one-step lookahead. The stability region is the set of all $\tilde{J}$ such that the policy $\tilde{\mu}$ such that $T\tilde{J} = T_{\tilde{\mu}}\tilde{J}$ satisfies $J_{\tilde{\mu}}(x) < \infty$ for all x.

holds, the region of stability includes the cost functions J_μ of all stable policies μ. In particular, *rollout with a stable policy provides a starting point J_μ for approximation in value space, which lies within the region of stability*.

Figure 3.6 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. 3.5. However, the figure indicates that *the region of stability of the ℓ-step lookahead controller $\tilde{\mu}$ depends on ℓ, and tends to increase 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). A conjecture that should hold under reasonable conditions is that if the VI algorithm $J_k = T^k\tilde{J}$ converges to J^* then $T^{\ell-1}\tilde{J}$ belongs to the region of stability for sufficiently large ℓ. We will revisit this issue in Section 3.5.

3.3 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. Each iteration of the PI algorithm starts with a policy (which we call *current* or *base* policy), and generates another policy (which we call *new* or *rollout* policy). For the infinite horizon problem

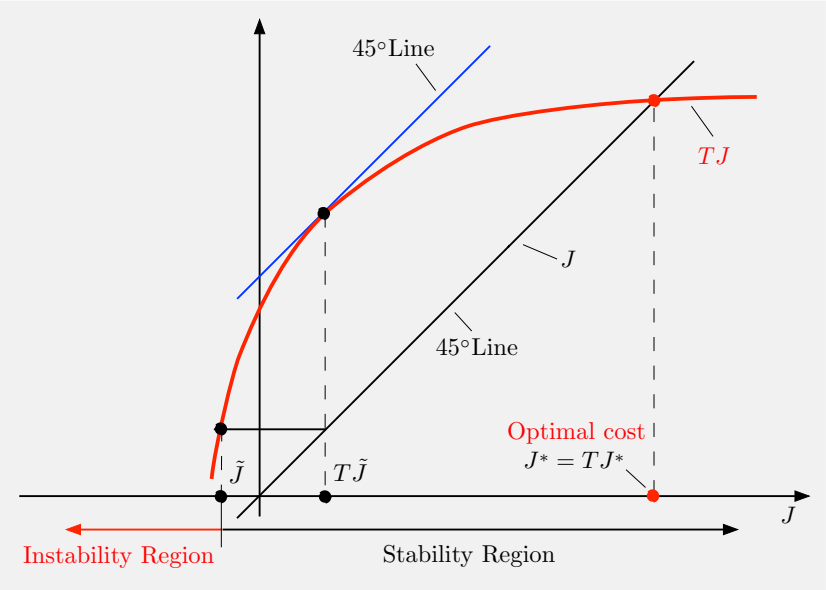

Figure 3.6 Illustration of the regions of stability and instability for approximation in value space with multistep lookahead. The stability region is the set of all $\tilde{J}$ for which the policy $\tilde{\mu}$ such that $T^\ell \tilde{J} = T_{\tilde{\mu}} T^{\ell-1} \tilde{J}$ satisfies $J_{\tilde{\mu}}(x) < \infty$ for all x (the figure shows the case $\ell = 2$).

of Section 2.1, given the base policy μ, the iteration consists of two phases:

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

$$J_\mu(x) = E\Big\{g\big(x, \mu(x), w\big) + \alpha J_\mu\big(f(x, \mu(x), w)\big)\Big\}, \qquad \text{for all } x. \tag{3.5}$$

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 rollout policy $\tilde{\mu}$ using the one-step lookahead minimization

$$\tilde{\mu}(x) \in \arg\min_{u \in U(x)} E\Big\{g(x, u, w) + \alpha J_\mu\big(f(x, u, w)\big)\Big\}, \qquad \text{for all } x. \tag{3.6}$$

It is generally expected 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 policies $\{\mu^k\}$, by obtaining μ^{k+1} through a policy improvement operation using J_{μ^k} in place of J_μ in Eq. (3.6), which is obtained through policy evaluation of the preceding policy μ^k using Eq. (3.5). It is well known that PI has solid convergence properties that hold even when the method is implemented in unconventional computing environments, involving asynchronous distributed computation, as shown in a series of papers by Bertsekas and Yu [BeY10], [BeY12], [YuB13].

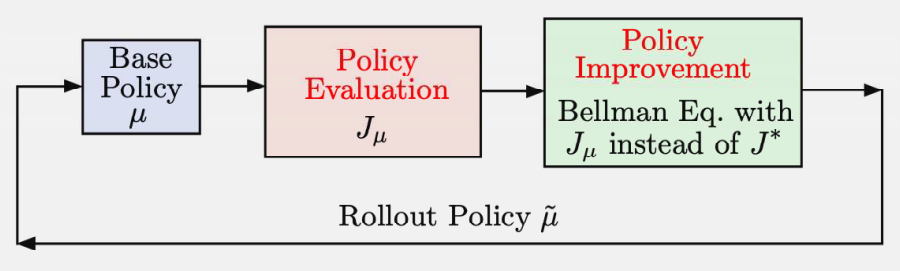

Figure 3.7 Schematic illustration of PI as repeated rollout. It generates a sequence of policies, with each policy μ in the sequence being the base policy that generates the next policy $\tilde{\mu}$ in the sequence as the corresponding rollout policy.

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

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k}, \tag{3.7}$$

while the policy improvement phase obtains μ^{k+1} through the equation

$$T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}; \tag{3.8}$$

see Fig. 3.8, which illustrates that PI can be viewed as Newton's method for solving the Bellman equation in the function space of cost functions J.

The interpretation of PI as a form of Newton's method has a long history, for which we refer to the original papers by Kleinman [Klei68] for linear quadratic problems, and by Pollatschek and Avi-Itzhak [PoA69] for the finite-state discounted and Markov game cases. Subsequent works, which address broader classes of problems and algorithmic variations, include (among others) Hewer [Hew71], Puterman and Brumelle [PuB78], [PuB79], Saridis and Lee [SaL79] (following Rekasius [Rek64]), Beard [Bea95], Beard, Saridis, and Wen [BSW99], Santos and Rust [SaR04], Bokanowski, Maroso, and Zidani [BMZ09], Hylla [Hyl11], Magirou, Vassalos, and Barakitis [MVB20], Bertsekas [Ber21c], and Kundu and Kunitsch [KuK21]. Some of these papers include superlinear convergence rate results.

Rollout

Generally, rollout with 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. 3.8). Because rollout is applied only once at each stage during the real-time operation of the system, it is well suited for on-line implementation, provided that the policy evaluation of the base policy can be done on-line as needed. This often involves deterministic or stochastic on-line simulation from each of the states x_k generated by the system in real time.

As Fig. 3.8 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

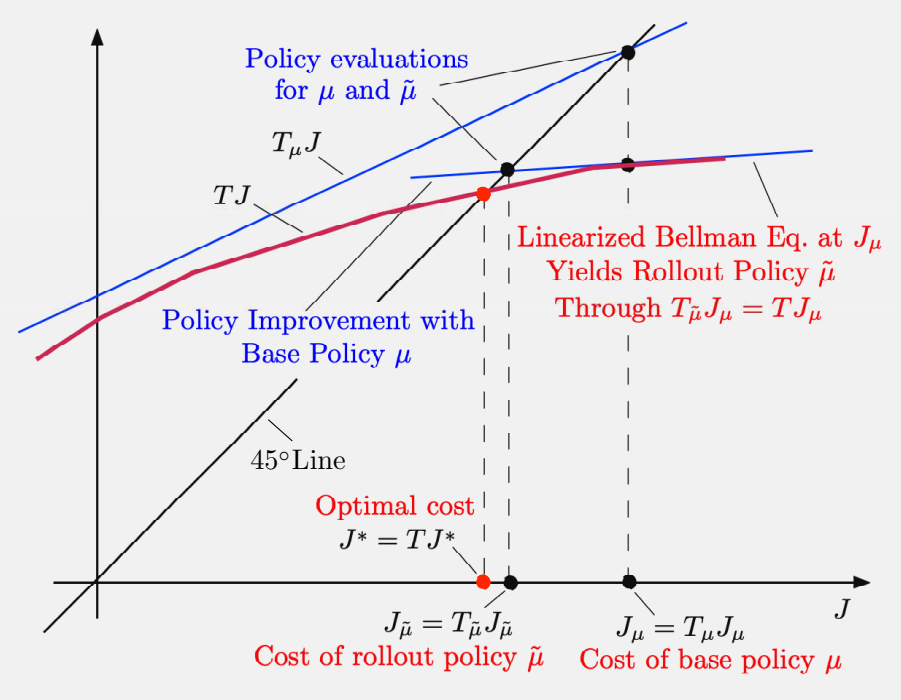

Figure 3.8 Geometric interpretation of PI and rollout. Each policy μ defines the linear function $T_\mu J$ of J, given by Eq. (3.2), and TJ is the function given by Eq. (3.1), which can also be written as $TJ = \min_\mu T_\mu J$. The figure shows a policy iteration starting from a base policy μ. It computes 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.

TJ is nearly linear (i.e., has small "curvature") the rollout policy performance $J_{\tilde{\mu}}(x)$ is very close to the optimal $J^*(x)$, even if the base policy μ is far from optimal. This explains the large cost improvements that are typically observed in practice with the rollout algorithm.

An interesting question is how to compare the rollout performance $J_{\tilde{\mu}}(x)$ for a given initial state x, with the base policy performance $J_\mu(x)$. Clearly, we would like $J_\mu(x) - J_{\tilde{\mu}}(x)$ to be large, but this is not the right way to look at cost improvement. The reason is that $J_\mu(x) - J_{\tilde{\mu}}(x)$ will be small if its upper bound, $J_\mu(x) - J^*(x)$, is small, i.e., if the base policy is close to optimal. What is important is that the error ratio

$$\frac{J_{\tilde{\mu}}(x) - J^*(x)}{J_\mu(x) - J^*(x)} \tag{3.9}$$

is small. Indeed, this ratio becomes smaller as $J_\mu(x) - J^*(x)$ approaches 0 because of the superlinear convergence rate of Newton's method that underlies the rollout algorithm (cf. Fig. 3.8). Unfortunately, it is hard to evaluate this ratio, since we do not know $J^*(x)$. On the other hand, we should not be underwhelmed if we observe a small performance improvement $J_\mu(x) - J_{\tilde{\mu}}(x)$: the reason may be that the base policy is already near-optimal, and in fact we may be doing very well in terms of the ratio (3.9).

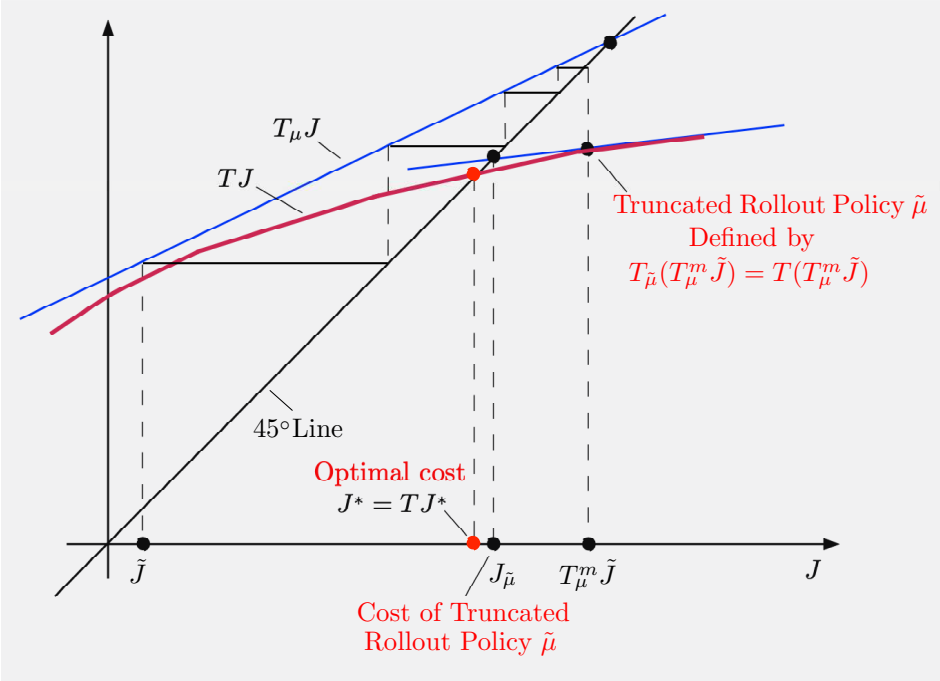

Figure 3.9 Geometric interpretation of truncated rollout that uses one-step lookahead minimization, m VIs with the base policy μ, and a terminal cost function approximation $\tilde{J}$ (in this figure $m = 4$).

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. 3.9. 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. This is closely connected to a variant of PI called *optimistic*; see [BeT96], [Ber12], [Ber19a] for a more detailed discussion of this relation. 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. 3.9. In the case of ℓ-step lookahead, the truncated rollout policy $\tilde{\mu}$ is defined by

$$T_{\tilde{\mu}}(T^{\ell-1} T_\mu^m \tilde{J}) = T(T^{\ell-1} T_\mu^m \tilde{J}).$$

An interesting observation from Fig. 3.9 is that it may not be beneficial to make m, the number of value iterations with μ, very large. This has often been confirmed in practice, in the experience of the author and other researchers.

As noted earlier, variants of Newton's method that involve multiple fixed point iterations, before and after each Newton step, but without rollout with a base policy, i.e.,

$$T_{\tilde{\mu}}(T^{\ell-1}\tilde{J}) = T(T^{\ell-1}\tilde{J}), \tag{3.10}$$

are called Newton-SOR, and are well-known. The numerical analysis book by Ortega and Rheinboldt [OrR70] (Sections 13.3 and 13.4) provides 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 (3.10), 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 optimistic policy iteration in DP; see e.g., [BeT96], [Ber12].

Geometrical interpretations such as the ones of Fig. 3.9 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 as an economic alternative to $(\ell + m)$-step lookahead minimization using $\tilde{J}$ as terminal cost function approximation.

The Effect of Implementation Errors in Approximation in Value Space

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 the discussion of multiagent problems in the next section). 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, adaptive, and model predictive control).

Under these circumstances the linearization of the Bellman equation at the point $\tilde{J}$ in Fig. 3.9 is perturbed, and the corresponding point $T_\mu^m \tilde{J}$ in Fig. 3.9 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 in the calculation of $T_\mu^m \tilde{J}$, the error in $J_{\tilde{\mu}}$ will be of the much smaller order $o(\epsilon)$, 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.

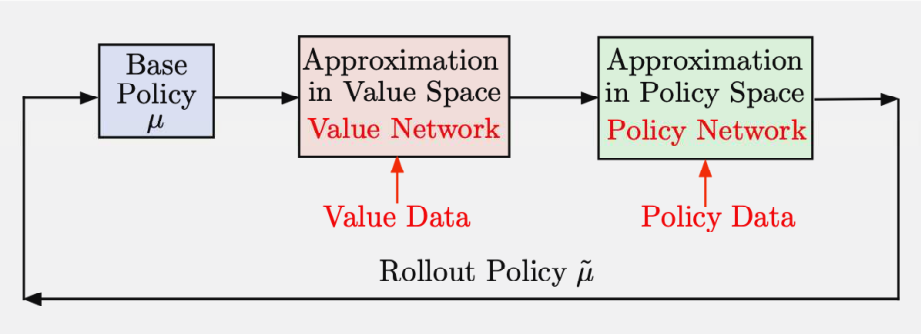

Figure 3.10 Schematic illustration of approximate PI. Either the policy evaluation and policy improvement phases (or both) are approximated with a value or a policy network, respectively. These could be neural networks, which are trained with (state, cost function value) data that is generated using the current base policy μ, and with (state, rollout policy control) data that is generated using the rollout policy $\tilde{\mu}$; see Chapters 4 and 5. Note that there are three different types of approximate implementation involving: 1) a value network but no policy network, or 2) a policy network but no value network, or 3) both a policy and a value network.

Approximate Policy Iteration and Implementation Errors

Both policy evaluation and policy improvement can be approximated, possibly using neural networks, and training with data collected through simulation; see Fig. 3.10. Other approximations include truncated rollout for policy evaluation. Moreover, multistep lookahead may be used in place of one-step lookahead, and simplified minimization, based for example on multiagent rollout, may also be used. Let us also mention the possibility of a combined rollout and PI algorithm, whereby we use PI for on-line policy improvement of the base policy, by using data collected during the rollout process. This idea is relatively new and has not been tested extensively; see the subsequent discussion in Section 5 and the author's paper [Ber21].

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 changes in 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 has little and unpredictable effect on the performance of the generated policies. A plausible conjecture here is that the superlinear convergence property of Newton's method "smooths out" the effect of the approximation errors.

3.4 Multiagent Problems and Multiagent Rollout

A major difficulty in the implementation of value space approximation with one-step lookahead is the minimization operation over $U(x_k)$. When $U(x_k)$ is infinite, or even when it is finite but has a very large number of elements, the minimization may become prohibitively time consuming. In the case of multistep lookahead the computational difficulty becomes even more acute. In this section we discuss how to deal with this

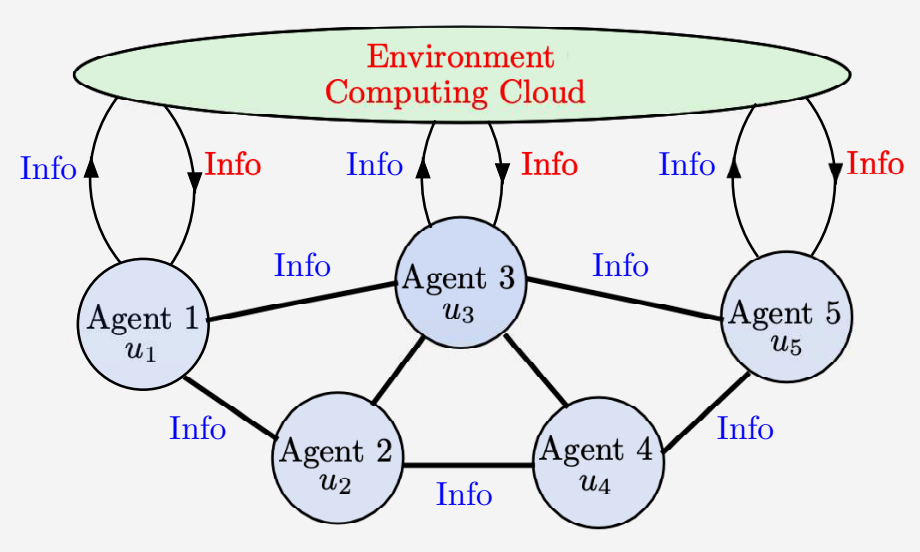

Figure 3.11 Schematic illustration of a multiagent problem. There are multiple "agents," and each agent $\ell = 1, \ldots, m$, controls its own decision variable u_ℓ. At each stage, agents exchange new information and also exchange information with the "environment," and then select their decision variables for the stage.

difficulty when the control u consists of m components, $u = (u_1, \ldots, u_m)$, with a separable control constraint for each component, $u_\ell \in U_\ell(x)$, $\ell = 1, \ldots, m$. Thus the control constraint set is the Cartesian product

$$U(x) = U_1(x) \times \cdots \times U_m(x), \tag{3.11}$$

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. 3.11.

To illustrate our approach, let us consider the discounted infinite horizon problem, and for the sake of the following discussion, 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

$$\tilde{u} \in \arg\min_{u \in U(x)} E\Big\{g(x,u,w) + \alpha J_\mu\big(f(x,u,w)\big)\Big\}, \tag{3.12}$$

and involves as many as n^m terms, 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 (3.11)]. Thus the standard rollout algorithm requires an exponential [order $O(n^m)$] number of 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 (3.12) 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

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

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

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

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

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 $O(nm)$ 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 (3.12) are proportional to the number of controls in the set $U_k(x_k)$ and are independent of the size of the state space. This motivates a reformulation of the problem, first suggested in the book [BeT96], Section 6.1.4, whereby control space complexity is traded off with state space complexity, by "unfolding" the control u_k into its m components, which are applied *one agent-at-a-time* rather than all-agents-at-once.

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

To this end, we introduce a modified but equivalent problem, involving one-at-a-time agent control selection. At the generic state x, we break down the control u into the sequence of the m controls $u_1,u_2,\ldots,u_m$, and between x and the next state $\bar{x}=f(x,u,w)$, we introduce artificial intermediate "states" $(x,u_1),(x,u_1,u_2),\ldots,(x,u_1,\ldots,u_{m-1})$, and corresponding transitions. The choice of the last control component u_m at "state" $(x,u_1,\ldots,u_{m-1})$ 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. 3.12.

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

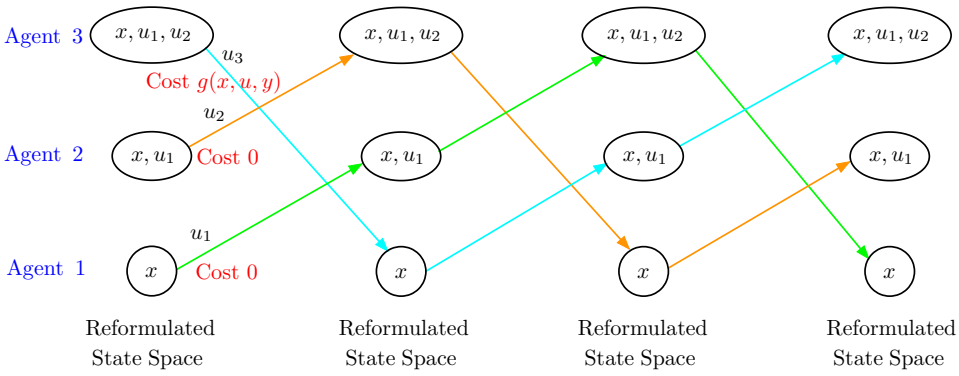

Figure 3.12 Equivalent formulation of the N-stage stochastic optimal control problem for the case where the control u consists of m components $u_1, u_2, \ldots, u_m$:

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

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.

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 minimization (3.12) is performed for just one state at each stage.

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

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

where $J_\mu(x)$ is the cost function of the base policy μ, and $J_{\tilde{\mu}}(x)$ is the cost function of the rollout policy $\tilde{\mu} = (\tilde{\mu}_1, \ldots, \tilde{\mu}_m)$, starting from state x. Furthermore, this cost improvement property can be extended to multiagent PI schemes that involve one-agent-at-a-time policy improvement operations, and have sound convergence properties (see the book [Ber20a], Chapters 3 and 5, as well as the author's papers [Ber19b], [Ber19c], [Ber20b], [Ber21b], and [BKB20]). Moreover, multiagent rollout can become the starting point for various related PI schemes that are well suited for distributed operation in important practical contexts involving multiple autonomous decision makers (see the book [Ber20a], Section 5.3.4, and the paper [Ber21b]).

3.5 Linear Quadratic Problem - Illustrations

In this section, we will use linear quadratic problems as a vehicle for insight into the suboptimal control ideas developed so far. This is possible because linear quadratic problems admit closed form solutions. Our discussion applies to multidimensional linear quadratic problems of the form (2.5)-(2.6), but we will focus on the one-dimensional case to demonstrate graphically the approximation in value space ideas of this section. In particular, we will consider the system

$$x_{k+1} = ax_k + bu_k, \tag{3.13}$$

and the cost function

$$\sum_{k=0}^{\infty}(qx_k^2 + ru_k^2), \tag{3.14}$$

where a, b, q, r are scalars with $b \neq 0$, $q > 0$, $r > 0$ (we will later allow q to take the value 0 as well).

The optimal solution of the problem is well known (see for example [Ber17a], Section 3.1). The optimal cost function has the form

$$J^*(x) = K^*x^2, \tag{3.15}$$

where the scalar K^* solves a fixed point equation of the form

$$K = F(K), \tag{3.16}$$

with F defined by

$$F(K) = \frac{a^2rK}{r + b^2K} + q. \tag{3.17}$$

This equation, known as the algebraic *Riccati equation*, is derived from (and is equivalent to) the Bellman equation $J = TJ$, restricted to the space of quadratic functions of the form $J(x) = Kx^2$. The optimal cost function J^* corresponds to K^*, which is the unique solution of the Riccati equation (3.16) within the positive real line (this equation has another solution within the negative real time, which is of no interest); see Fig. 3.13.†

The optimal policy is a linear function of the state and has the form

$$\mu^*(x) = L^*x, \tag{3.18}$$

where L^* is the scalar given by

$$L^* = -\frac{abK^*}{r + b^2K^*}. \tag{3.19}$$

† For multidimensional problems, the Riccati equation involves symmetric matrices K, and has a unique solution within the class of positive semidefinite matrices under well-known assumptions of controllability and observability; see [Ber17a], Section 3.1.

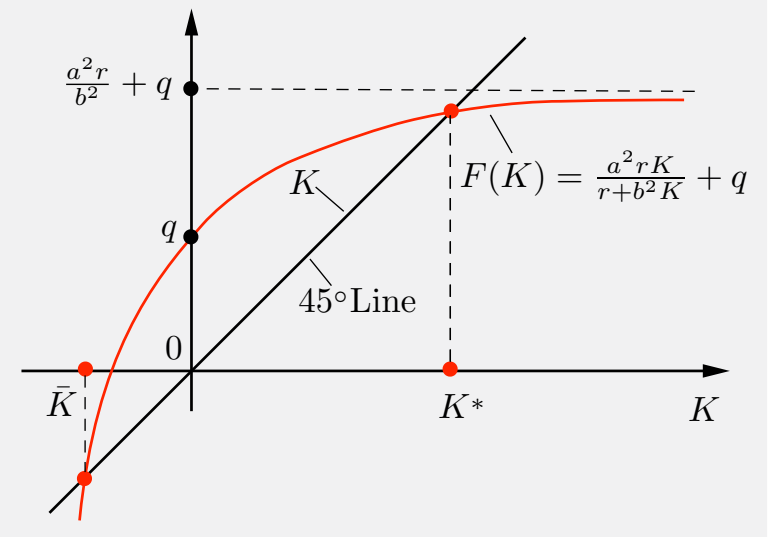

Figure 3.13 Graphical construction of the solutions of the Riccati equation for the linear quadratic problem, starting from a quadratic initial function. The optimal cost function is $J^*(x) = K^*x^2$, where the scalar K^* solves the fixed point equation $K = F(K)$, where F is the function given by

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

Because 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 equation $K = F(K)$ (known as the Riccati equation) has one positive solution K^* and one negative solution, denoted $\bar{K}$.

Moreover, given any linear policy of the form

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

where L is a scalar, the corresponding closed loop system is

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

and the cost $J_\mu(x_0)$ is calculated as

$$\sum_{k=0}^{\infty} \left(q(a + bL)^{2k}x_0^2 + rL^2(a + bL)^{2k}x_0^2\right) = \lim_{N\to\infty} \sum_{k=0}^{N-1} (q + rL^2)(a + bL)^{2k}x_0^2,$$

Assuming $|a + bL| < 1$, i.e., that the closed loop system is stable, the above summation yields

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

for every initial state x, where

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

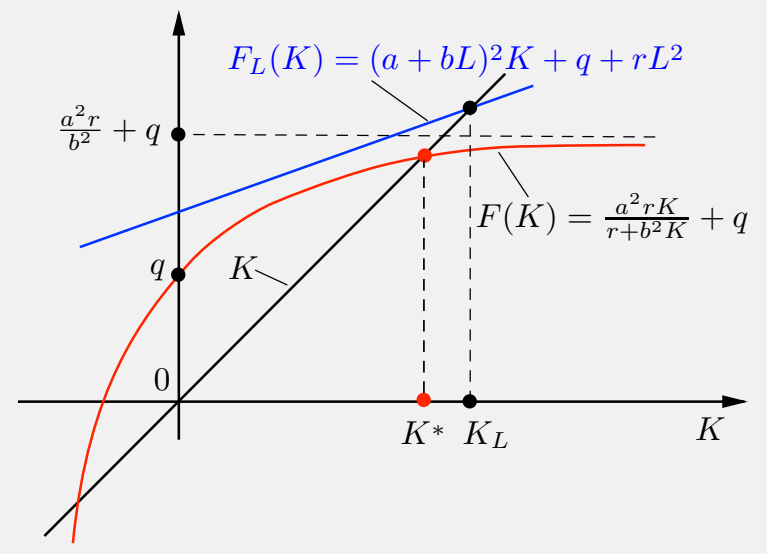

Figure 3.14 Illustration of the construction of the cost function of a stable linear policy $\mu(x) = Lx$. The cost function $J_\mu(x)$ has the form

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

where K_L is the unique solution of the linear equation $K = F_L(K)$, where

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

If μ is unstable, we have $J_\mu(x) = \infty$ for $x \neq 0$.

With a straightforward calculation it can be verified that K_L is the unique solution of the linear equation

$$K = F_L(K),$$

where

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

see Fig. 3.14. This is equivalent to the Bellman equation $J = T_\mu J$ for the policy μ. On the other hand when $|a+bL| > 1$, and the system is unstable, we have $J_\mu(x) = \infty$ for all $x \neq 0$.

The one-dimensional problem of this section is well suited for geometric interpretations such as the ones we gave earlier in this section, because approximation in value space, and the VI, rollout, and PI algorithms, involve quadratic cost functions $J(x) = Kx^2$, which can be represented by one-dimensional graphs as functions of just the number K. In particular, Bellman's equation can be replaced by the Riccati equation (3.17). Similarly, the approximation in value space with one-step and multistep lookahead Figs. 3.3-3.4, the region of stability Figs. 3.5-3.6, and the rollout and PI Figs. 3.8-3.9 can be represented by one-dimensional graphs. We will next present these graphs and obtain corresponding geometrical insights. In

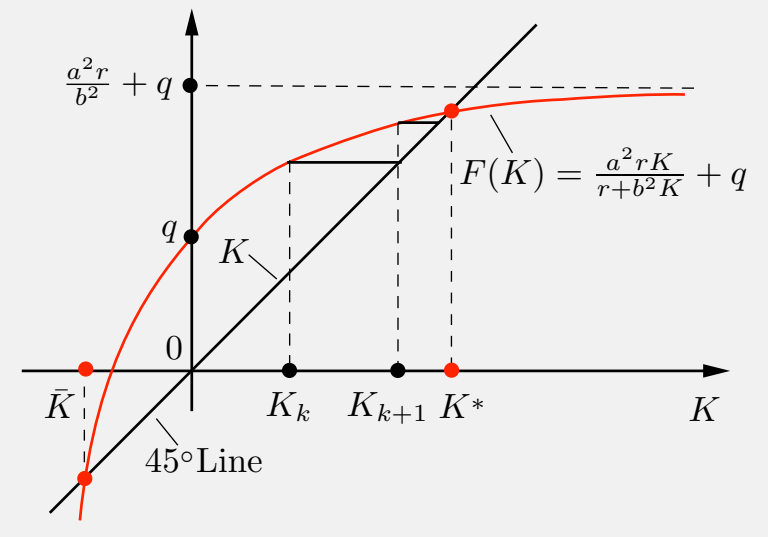

Figure 3.15 Illustration of the VI algorithm for the linear quadratic problem. It has the form $K_{k+1} = F(K_k)$ where

$$F(K) = \frac{a62rK}{r + b^2K} + q.$$

It is essentially equivalent to the VI algorithm with a quadratic starting function $J_0(x) = K_0x^2$. The algorithm converges to K^* starting from anywhere in the interval $(\bar{K}, \infty)$, where $\bar{K}$ is the negative solution, as shown in the figure. Starting from values K_0 with $-r/b^2 < K_0 \leq \bar{K}$, the algorithm converges to the negative solution $\bar{K}$.

Section 3.5, we will also obtain similar insights about what happens in exceptional cases, including the case where we have $q = 0$. Note that our discussion applies qualitatively to multidimensional linear quadratic problems of the form (2.5)-(2.6), and can be verified to a great extent by analysis, but an effective geometrical illustration is only possible when the system is one-dimensional.

Bellman's Equation and Value Iteration

The VI algorithm for the one-dimensional linear quadratic problem is illustrated in Fig. 3.15. It can be shown, and can also be seen from the figure, that the algorithm, which has the form

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

converges to K^* starting from anywhere in the interval $(\bar{K}, \infty)$, where $\bar{K}$ is the negative solution. In particular, the algorithm converges to K^* starting from any nonnegative value of K. It is interesting to note that, starting from values K_0 with $-r/b^2 < K_0 \leq \bar{K}$, the algorithm converges to the negative solution $\bar{K}$. When $K_0 \leq -r/b^2$, for the corresponding function $J(x) = K_0x^2$ we have $(TJ)(x) = -\infty$ for all x, and the algorithm is undefined. The literature on linear quadratic problems universally assumes that the iterative solution of the Riccati equation is started with nonnegative K_0, since to select a negative starting K_0 makes little sense: it slows down the convergence of the iteration without offering any apparent advantage.

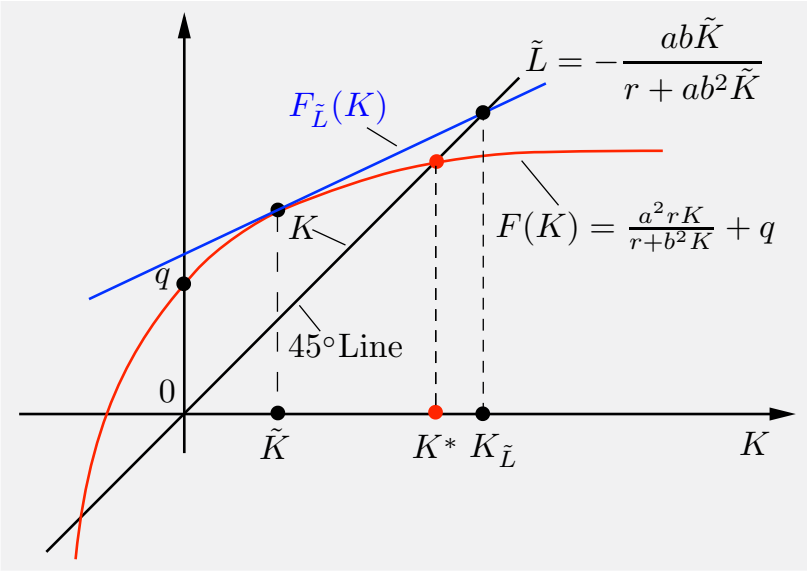

Figure 3.16 Illustration of approximation in value space with one-step lookahead for the linear quadratic problem. Given a terminal cost approximation $\tilde{J} = \tilde{K}x^2$, we compute the corresponding linear policy $\tilde{\mu}(x) = \tilde{L}x$, where

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

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

Approximation in Value Space with One-Step and Multistep Lookahead

Approximation in value space with a quadratic terminal cost approximation $\tilde{L}(x) = \tilde{K}x^2$ is easily implemented because the one-step or multistep lookahead policy can be obtained analytically as a linear policy. In particular, for the one-step lookahead minimization, we have

$$\tilde{\mu}(x) \in \arg\min_u \left[qx^2 + ru^2 + \tilde{K}(ax + bu)^2\right],$$

which after a straightforward calculation, yields

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

with the linear policy coefficient given by

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

We may also construct the linearization of the function F at $\tilde{K}$, and solve the corresponding linearized problem with a Newton step, as illustrated in Fig. 3.16. The case of ℓ-step lookahead minimization can be similarly interpreted. 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 $\tilde{K}$. Figure 3.17 depicts the case $\ell = 2$.

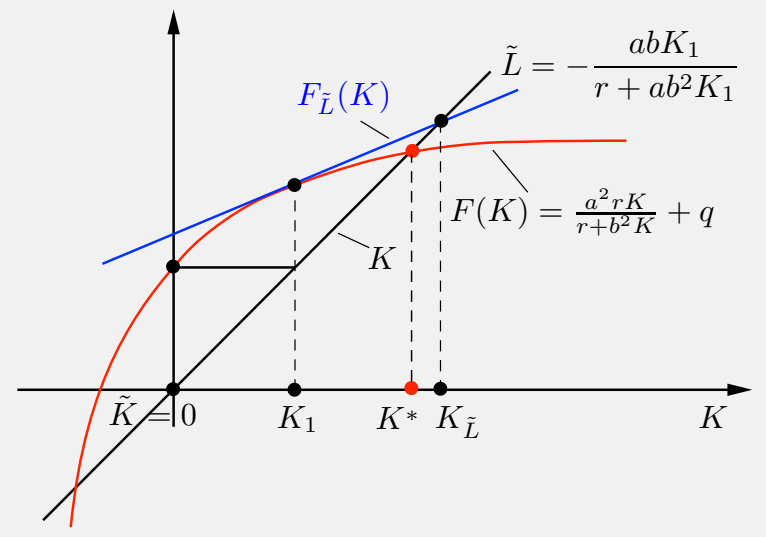

Figure 3.17 Illustration of approximation in value space with two-step lookahead for the linear quadratic problem. Starting with a terminal cost approximation $\tilde{J} = \tilde{K}x^2$ ($\tilde{K} = 0$ in the figure), we obtain K_1 using a single value iteration. We then compute the corresponding linear policy $\tilde{\mu}(x) = \tilde{L}x$, where

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

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

Rollout and Policy Iteration

Let us derive the PI algorithm starting from a linear base policy of the form

$$\mu^0(x) = L_0x,$$

where L_0 is a scalar such that the closed loop system

$$x_{k+1} = (a + bL_0)x_k, \tag{3.20}$$

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

To describe the policy evaluation and policy improvement phases for the starting policy μ^0, we first calculate J_{μ^0} by noting that it involves the uncontrolled closed loop system (3.20) and a quadratic cost function. Similar to our earlier calculations, it has the form

$$J_{\mu^0}(x) = K_0x^2, \tag{3.21}$$

where

$$K_0 = \frac{q + rL_0^2}{1 - (a + bL_0)^2}. \tag{3.22}$$

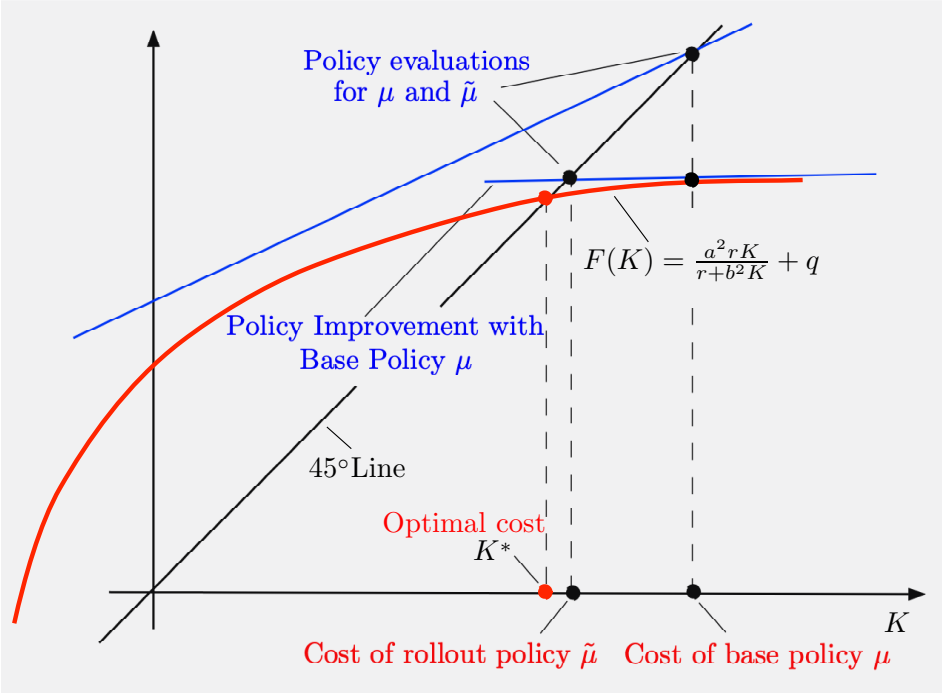

Figure 3.18 Illustration of rollout and policy iteration for the linear quadratic problem.

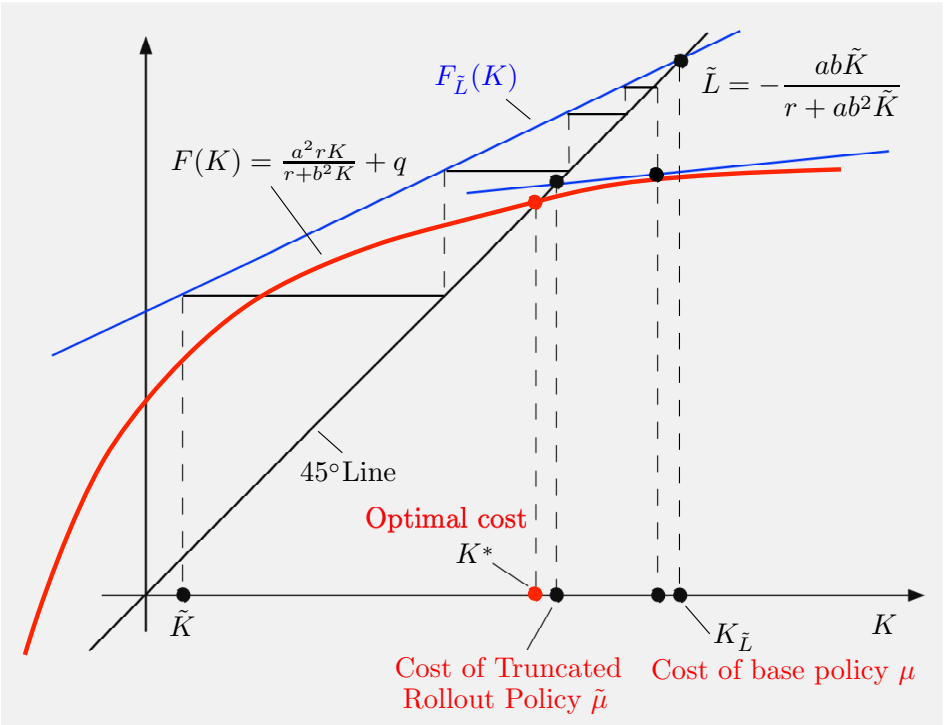

Figure 3.19 Illustration of truncated rollout and optimistic policy iteration for the linear quadratic problem.

Thus, the policy evaluation phase of PI for the starting linear policy $\mu^0(x) = L_0 x$ yields J_{μ^0} in the form (3.21)-(3.22). The policy improvement phase involves the quadratic minimization

$$\mu^1(x) \in \arg\min_u \big[qx^2 + ru^2 + K_0(ax + bu)^2\big],$$

and after a straightforward calculation yields μ^1 as the linear policy $\mu^1(x) = L_1 x$, where

$$L_1 = -\frac{abK_0}{r + ab^2 K_0}.$$

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

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

$$\mu^k(x) = L_k x, \qquad k = 0, 1, \ldots, \tag{3.23}$$

where L_k is generated by the iteration

$$L_{k+1} = -\frac{\alpha b K_k}{r + \alpha b^2 K_k}, \tag{3.24}$$

with K_k given by

$$K_k = \frac{q + rL_k^2}{1 - (a + bL_k)^2}, \tag{3.25}$$

[cf. Eq. (3.22)]. The corresponding cost function sequence has the form $J_{\mu^k}(x) = K_k x^2$ for the deterministic problem where $w_k \equiv 0$, and the form

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

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

3.6 Exceptional Cases

Let us now consider situations where exceptional behavior occurs. One such situation is when the Bellman equation $J = TJ$ has multiple solutions. Then the VI algorithm, when started at one of these solutions will stay at that solution. More generally, it may be unclear whether the VI algorithm with converge to J^*, even when started at seemingly favorable initial conditions.

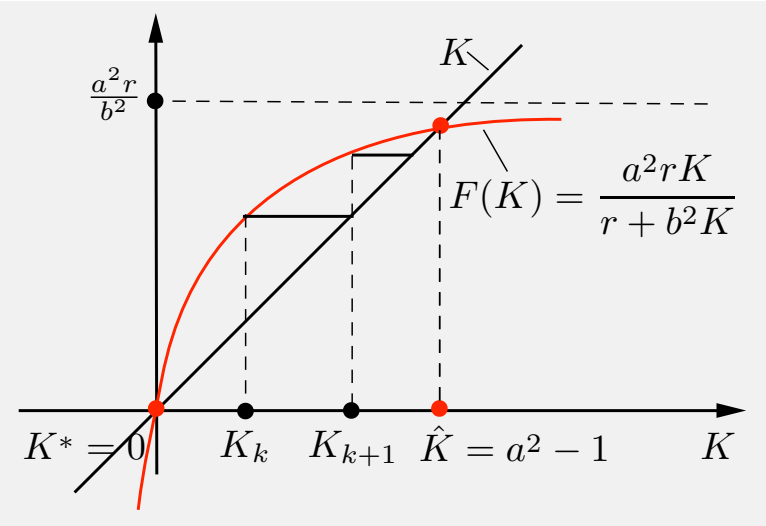

Figure 3.20 Illustration of the Bellman equation and the VI algorithm for the linear quadratic problem in the exceptional case where $q = 0$.

A Linear Quadratic Problem

It turns out that such exceptional behavior can occur for the one-dimensional linear quadratic problem when $q = 0$ and the system is unstable when left uncontrolled, i.e., $a > 1$; see Fig. 3.20. In this case, since the cost does not depend on the state, it is optimal to apply control $u = 0$ at any state x, i.e., $\mu^*(x) \equiv 0$, and the optimal cost function is $J^*(x) \equiv 0$. Then the Riccati equation has two solutions:

$$K^* = 0 \qquad \text{and} \qquad \hat{K} = a^2 - 1,$$

as shown in Fig. 3.20. The solution K^* corresponds to the optimal cost function. It turns out that the solution $\hat{K}$ has an interesting interpretation: it can be shown to be *the optimal cost function within the subclass of linear policies that are stable*; a proof of this is given in the author's paper [Ber17a], and the monograph [Ber18a], Section 3.1.

Consider also the VI algorithm starting from some $K_0 > 0$. As shown from Fig. 3.20, it generates a positive scalar sequence that converges to $\hat{K}$. If the VI algorithm is started at the optimal $K^* = 0$, it it stays at K^*. It can also be verified that the PI algorithm generates a sequence of linear stable policies starting from a linear stable policy. The sequence converges to the optimal stable policy that corresponds to $\hat{K}$. In summary, the PI algorithm starting from a linear stable policy converges to $\hat{J}$, the optimal cost function over linear stable policies, but not to the optimal policy.

Regarding rollout, it is essential to use a stable base policy, for otherwise the cost function of the base policy is infinite for some states x, and the rollout policy cannot be defined. Finally, the region of stability is of the form $(\tilde{K}, \infty)$, where $\tilde{K}$ is some positive critical level, similar to the standard case where $q > 0$. In

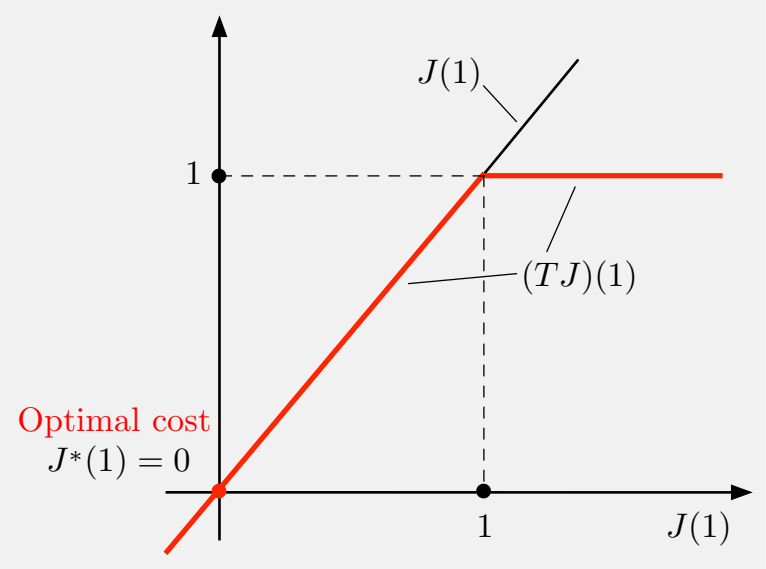

Figure 3.21 Illustration of the Bellman equation for a shortest path problem in the exceptional case where there is a cycle of zero length. Restricted within the set of J with $J(t) = 0$, the Bellman operator has the form

$$(TJ)(1) = \min\big\{J(1),\, 1\big\}.$$

The set of solutions of Bellman's equation, $J(1) = (TJ)(1)$ is the interval $(-\infty, 1]$ and contains $J^*(1) = 0$ in its interior.

fact $\tilde{K}$ is such that the derivative $dF(K)/dK$ is equal to 1. By solving the equation

$$\frac{dF(K)}{dK} = \frac{a^2 r^2}{(r + b^2 K)^2} = 1,$$

we obtain

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

Thus, when the system is unstable if left uncontrolled ($a > 1$), to obtain a stable system by one-step lookahead, we must use a cost function approximation that is strictly positive and exceeds $\tilde{K}x^2$.

A Shortest Path Problem

Another exceptional type of example is provided by shortest path problems that contain cycles of zero length; see the monograph [Ber18a], Section 3.1. In this case there are infinitely many solutions to Bellman's equation, and the VI and the PI algorithm, as well as the approximation in value space process exhibit unusual behavior. We demonstrate this with a shortest path problem involving a single state 1, in addition to the cost-free destination state t.

In particular, let $X = \{t, 1\}$, and assume that at state 1 there are two options: we can stay at 1 at cost 0, or move to t at cost 1. Here $J^*(t) = J^*(1) = 0$, and there are just two policies that correspond to

the two options at state 1. The optimal policy starting at state 1 is to stay at 1. If we restrict attention to cost functions J with $J(t) = 0$, the Bellman operator is

$$(TJ)(1) = \min\big\{J(1),\, 1\big\},$$

and Bellman's equation, written as an equation in $J(1)$, has the form

$$J(1) = \min\big\{J(1),\, 1\big\}.$$

The set of solutions of this equation is the interval $(-\infty, 1]$ and it is infinite; see Fig. 3.21. The optimal value $J^*(1) = 0$ lies in the interior of this set, and cannot be obtained by the VI algorithm, unless the algorithm is started at the optimal value.

Let us consider approximation in value space with cost approximation $\tilde{J}(1)$. Then it can be seen that if $\tilde{J}(1) < 1$, the one-step lookahead policy is to stay at state 1, which is optimal. If $\tilde{J}(1) > 1$, the one-step lookahead policy is to move from state 1 to state t, which is suboptimal. If $\tilde{J}(1) = 1$, either one of the two policies can be the one-step lookahead policy.

Consider also the PI algorithm, starting from the suboptimal policy μ that moves from state 1 to state t. Then $J_\mu(t) = 0$, $J_\mu(1) = 1$, and it can be seen that μ satisfies the policy improvement equation

$$\mu(1) \in \arg\min\big\{J_\mu(t),\, 1 + J_\mu(1)\big\},$$

(the same is true for the optimal policy that stays at state 1). Thus the PI algorithm may stop with the suboptimal policy μ.

4. ROBUSTNESS ISSUES

Our discussion so far dealt with problems with a known and unchanging 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. From our point of view, it makes no difference whether the mathematical model is available through closed form mathematical expressions or through a computer simulator: the methods that we discuss are valid either way, only their suitability for a given problem may be affected by the availability of mathematical formulas.

In practice, however, it is common that the system involves parameters that are either not known exactly or may change over time. In such cases it is important to design controllers that take the parameter

changes into account. The methodology for doing so is generally known as *adaptive control*, an intricate and multifaceted subject, with many and diverse applications, and a long history.†

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

In what follows in this section, we will briefly review some of the principal types of adaptive control methods. We will then focus on schemes that are based on on-line replanning, including the use of rollout.

Robust and PID Control

Given a controller design that has been obtained assuming a nominal DP problem model, one possibility is to simply ignore changes in problem parameters. We may then try to design a controller that is adequate for the entire range of the changing parameters. This is sometimes called a *robust controller*. A robust controller makes no effort to keep track of changing problem parameters. It is just designed so that it is resilient to parameter changes.

An important time-honored robust control approach for continuous-state problems is the *PID (Proportional-Integral-Derivative) controller*; see e.g., the books by Aström and Hagglund [AsH95], [AsH06]. 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.

† The difficulties of designing adaptive controllers are often underestimated. Among others, they complicate the balance between off-line training and on-line play, which we discussed in Section 1 in connection to AlphaZero. It is worth keeping in mind that as much as learning to play high quality chess is a great challenge, the rules of play are stable and do not change unpredictably in the middle of a game! Problems with changing system parameters can be far more challenging!

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

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

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

This alternation of estimation and control phases may be repeated several times during the system's operation in order to take into account subsequent changes of the parameters. Note that it is not necessary to introduce a hard separation between the identification and the control phases. They may be going on simultaneously, with new parameter estimates being generated in the background, and introduced into the control process, whenever this is thought to be desirable; see Fig. 4.1.

One drawback of this approach is that it is not always easy to determine when to terminate one phase and start the other. A second difficulty, of a more fundamental nature, is that the control process may make some of the unknown parameters invisible to the estimation process. This is known as the problem of *parameter identifiability*, which is discussed in the context of adaptive control in several sources. On-line parameter estimation algorithms, which address among others the issue of identifiability, have been discussed extensively in the control theory literature, but the corresponding methodology is complex and beyond our scope in this book. However, assuming that we can make the estimation phase work somehow, we are free to reoptimize the controller using the newly estimated parameters, in a form of on-line replanning process.

Unfortunately, there is still another difficulty with this type of on-line replanning: it may be hard to recompute an optimal or near-optimal policy on-line, using a newly identified system model. In particular, it may be impossible to use time-consuming methods that involve for example the training of a neural network,

† In the adaptive control literature, schemes that involve parameter estimation are sometimes called *indirect*, while schemes that do not involve parameter estimation (like PID control) are called *direct*. To quote from the book by Aström and Wittenmark [AsW08], "indirect methods are those in which the estimated parameters are used to calculate required controller parameters" (see Fig. 4.1). The methods subsequently described in this section, and the rollout-based adaptive control methods discussed in the next section should be viewed as indirect. For accounts of adaptive control, we refer to the books by Bodson [Bod20], Goodwin and Sin [GoS84], Ioannou and Sun [IoS96], Jiang and Jiang [JiJ17], Krstic, Kanellakopoulos, and Kokotovic [KKK95], Kokotovic [Kok91], Kumar and Varaiya [KuV86], Liu, et al. [LWW17], Lavretsky and Wise [LaW13], Narendra and Annaswamy [NaA12], Sastry and Bodson [SaB11], Slotine and Li [SlL91], and Vrabie, Vamvoudakis, and Lewis [VVL13].

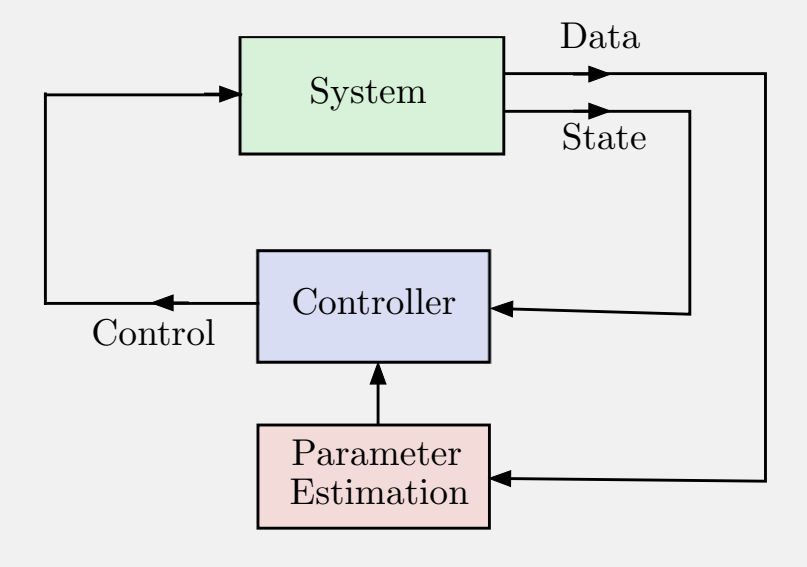

Figure 4.1 Schematic illustration of concurrent parameter estimation and system control. The system parameters are estimated on-line and the estimates are passed on to the controller whenever this is desirable (e.g., after the estimates change substantially). This structure is also known as indirect adaptive control.

or discrete/integer control constraints.† A simpler possibility is to use rollout, which we discuss in the next section.

4.1 Approximation in Value Space, Rollout, and Adaptive Control

We will now consider an approach for dealing with unknown or changing parameters, which is based on rollout and on-line replanning. We have already noted this approach in Section 1, where we stressed the importance of fast on-line policy improvement.

Let us assume that some problem parameters change over time and the controller becomes aware of the changes, perhaps after a suitable delay for data collection and estimation. The method by which the problem parameters are recalculated or become known is immaterial for the purposes of the following discussion. It may involve a limited form of parameter estimation, whereby the unknown parameters are "tracked" by data collection over a few time stages, with due attention paid to issues of parameter identifiability; or it

† Another possibility is to deal with this difficulty by precomputation. In particular, assume that the set of problem parameters may take a known finite set of values (or example each set of parameter values may correspond to a distinct maneuver of a vehicle, motion of a robotic arm, flying regime of an aircraft, etc). Then we may precompute a separate controller for each of these values. Once the control scheme detects a change in problem parameters, it switches to the corresponding predesigned current controller. This is sometimes called a *multiple model control design* or *gain scheduling*, and has been applied with success in various settings over the years.

may involve new features of the control environment, such as a changing number of servers and/or tasks in a service system.

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

Here the advantage of rollout is that it is simple, reliable, and relatively fast. In particular, it does not require a complicated training procedure, based for example on the use of neural networks or other approximation architectures, so *no new policy is explicitly computed in response to the parameter changes*. Instead the available controls at the current state are compared by a one-step or multistep minimization, with cost function approximation provided by the base policy (cf. Fig. 4.2).

Another issue to consider is the stability and robustness properties of the rollout policy. In this connection, it can be generally proved, under mild conditions, that *if the base policy is stable within a range of parameter values, the same is true for the rollout policy*; this can also be inferred from Fig. 3.8. Related ideas have a long history in the control theory literature; see Beard [Bea95], Beard, Saridis, and Wen [BSW99], Jiang and Jiang [JiJ17], Kalise, Kundu, Kunisch [KKK20].

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

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

The following example illustrates on-line replanning with the use of rollout in the context of the simple one-dimensional linear quadratic problem that we discussed earlier in this chapter. The purpose of the example is to show analytically how rollout with a base policy that is optimal for a nominal set of problem parameters works well when the parameters change from their nominal values. This property is not practically useful in linear quadratic problems because when the parameter change, it is possible to calculate the new optimal policy in closed form, but it is indicative of the performance robustness of rollout in other contexts; for example linear quadratic problems with constraints.

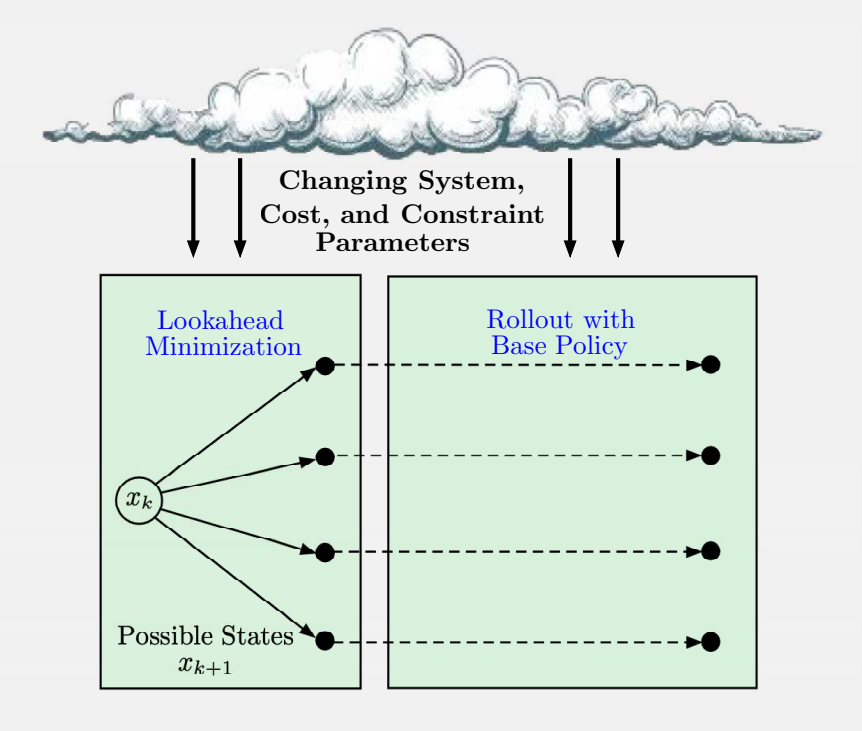

Figure 4.2 Schematic illustration of adaptive control by on-line replanning based on rollout. One-step lookahead is followed by simulation with the base policy, which stays fixed. The system, cost, and constraint parameters are changing over time, and the most recent estimates of their values are incorporated into the lookahead minimization and rollout operations. Truncated rollout with terminal cost approximation is also possible. For the discussion in this section, we may assume that all the changing parameter information is provided by some computation and sensor "cloud" that is beyond our control. The base policy may also be revised based on various criteria.

Example 4.1 (On-Line Replanning for Linear Quadratic Problems Based on Rollout)

Consider a deterministic undiscounted infinite horizon linear quadratic problem involving the linear system

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

and the quadratic cost function

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

This is the one-dimensional problem of the preceding section for the special case where $a = 1$ and $q = 1$. The optimal cost function is given by

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

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

$$K = \frac{rK}{r + b^2 K} + 1. \tag{4.1}$$

The optimal policy has the form

$$\mu^*(x) = L^* x, \tag{4.2}$$

where

$$L^* = -\frac{bK^*}{r + b^2K^*}. \tag{4.3}$$

As an example, consider the optimal policy that corresponds to the nominal problem parameters $b = 2$ and $r = 0.5$: this is the policy (4.2)-(4.3), with K obtained as the positive solution of the quadratic Riccati Eq. (4.1) for $b = 2$ and $r = 0.5$. We thus obtain

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

From Eq. (4.3) we then have

$$L = -\frac{2 + \sqrt{6}}{5 + 2\sqrt{6}}. \tag{4.4}$$

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. (4.1).

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

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

where L is given by Eq. (4.4). Here, we have

$$K_L = \frac{1 + rL^2}{1 - (1 + bL)^2}. \tag{4.5}$$

(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 derived earlier, we have

$$\tilde{L} = -\frac{bK_L}{r + b^2K_L}, \tag{4.6}$$

and

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

Figure 4.3 shows the coefficients K^*, K_L, and $\tilde{K}_L$ for a range of values of r and b. We have

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

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

Note that Fig. 4.3 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.

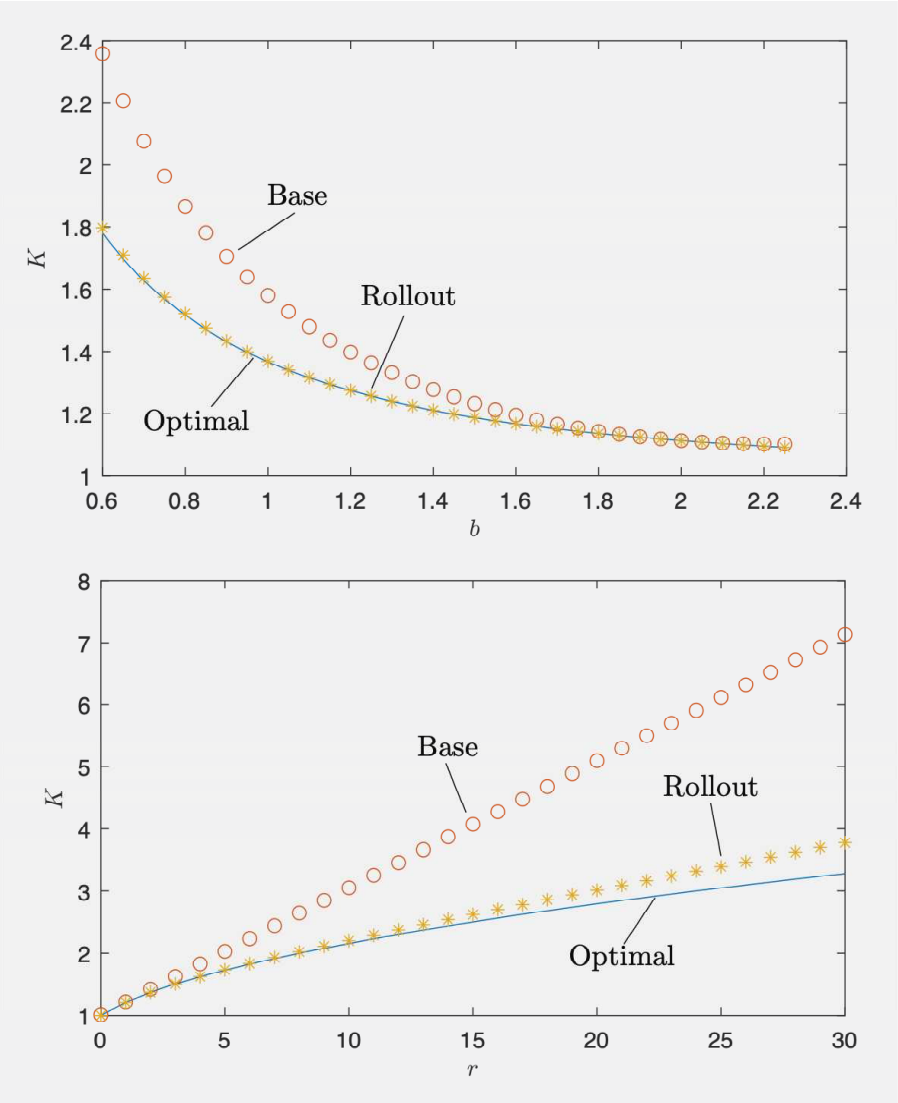

Figure 4.3 Illustration of control by rollout under changing problem parameters. The quadratic cost coefficients K^* (optimal, denoted by solid line), K_L (base policy, denoted by circles), and $\tilde{K}_L$ (rollout policy, denoted by asterisks) for the two cases where $r = 0.5$ and b varies, and $b = 2$ and r varies. The value of L is fixed at the value that is optimal for $b = 2$ and $r = 0.5$ [cf. Eq. (4.4)]. The rollout policy performance is very close to optimal, even when the base policy is far from optimal.

Note that, as the figure illustrates, we have

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

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

4.2 Approximation in Value Space, Rollout, and Model Predictive Control

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

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

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

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

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

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

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

The MPC algorithm at each encountered state x_k applies a control that is computed as follows; see Fig. 4.4:

(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

$$\min_{u_t,\, t=k,\ldots,k+\ell-1} \sum_{t=k}^{k+\ell-1} g(x_t, u_t), \tag{4.7}$$

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 $\tilde{u}_{k+1}, \ldots, \tilde{u}_{k+\ell-1}$.

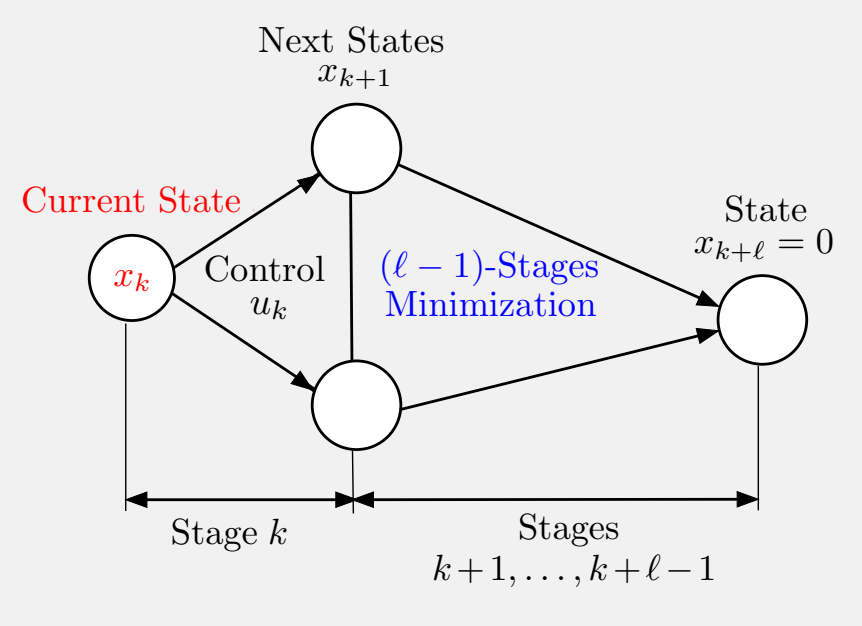

Figure 4.4 Illustration of the problem solved by MPC at state x_k. We minimize the cost function over the next ℓ stages while imposing the requirement that $x_{k+\ell}=0$. We then apply the first control of the optimizing sequence. In the context of rollout, the minimization over u_k is the one-step lookahead, while the minimization over $u_{k+1},\ldots,u_{k+\ell-1}$ that drives $x_{k+\ell}$ to 0 is the base heuristic.

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

The literature on MPC is voluminous, and for detailed accounts, we refer to the textbooks by 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]. A lot of this literature deals also with additional state constraints of the form $x_k \in X$, and addresses the issue of maintaining the state within X not just at the present time but also in future times. This is connected to the theory of reachability of target tubes, first studied by the author in his Ph.D. thesis [Ber71], and subsequent papers [BeR71], [Ber72]. According to this theory, to be able to satisfy for all k a state constraint of the form $x_k \in X$ using controls $u_k \in U(x_k)$, one needs to keep the state of the system within a smaller set $\bar{X} \subset X$ at all times. The calculation of $\bar{X}$ must be done off-line, and is part of the off-line training process. We do not discuss this issue and refer to the related MPC literature; see e.g., the book by Rawlings, Mayne, and Diehl [RMD17], the author's book [Ber20a], and the survey paper by Mayne [May14].

To make the connection of MPC with rollout, we note that *the one-step lookahead function $\tilde{J}$ implicitly used by MPC [cf. Eq. (4.7)] 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 [Ber05], is useful for making a connection with the approximate DP/RL and rollout in particular.

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.

There are several variants of MPC, which include the treatment of state constraints and the presence of uncertainty; see the discussion in the author's RL book [Ber20a], the MPC book by Rawlings, Mayne, and Diehl [RMD17], and the papers by Rosolia and Borelli [RoB17], [RoB19]. The paper by Li et al. [LJM21], instead of driving the state to 0 at the end of ℓ steps, considers multiple terminal system states at the end of the ℓ-step horizon, and the use of multiple base policies, which are obtained by off-line training.

In a common variant of MPC, the requirement of driving the system state to 0 in ℓ steps in the ℓ-stage MPC problem (4.7), 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 (4.7) 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

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

It can also be viewed as approximation in value space with ℓ-step lookahead and terminal cost approximation given by G. Thus our discussion of Section 3 relating to the region of stability of such schemes applies, and provides results that are known within the MPC framework under various conditions (see the paper by Mayne at al. [MRR00], and the MPC book [RMD17]).

Note that the preceding MPC controller may outperform substantially its base heuristic (in relative terms), particularly if the base heuristic is close to optimal [which is true if $G(x_{k+\ell}) \approx J^*(x_{k+\ell})+$ a constant]. This is due to the superlinear/quadratic convergence rate of Newton's method that underlies approximation in value space, as we have discussed in Section 3.

5. ON-LINE SIMPLIFIED POLICY ITERATION

In this section, we describe some variants of the PI algorithm, which were introduced in the author's recent paper [Ber21] for finite-state discounted infinite horizon DP problems, and are consistent with the approximation in value space theme of this paper. The salient feature of these variants is that *they are suitable for on-line implementation*. They are also simplified relative to standard PI in two ways:

(a) They perform policy improvement operations only for the states that are encountered during the on-line operation of the system.

(b) The policy improvement operation is simplified in that it uses approximate minimization over the Q-factors of the current policy at the current state.

Despite these simplifications, we show that our algorithms generate a sequence of improved policies, which converge to a policy with a local optimality property. Moreover, with an enhancement of the policy improvement operation, which involves a form of exploration, they converge to a globally optimal policy.

The motivation comes from the rollout algorithm, which starts from some available base policy and implements on-line an improved rollout policy. In the algorithm of the present section, the data accumulated from the rollout implementation are used to improve on-line the base policy, and to asymptotically obtain a policy that is either locally or globally optimal.

We assume a discrete-time dynamic system with states $1, \ldots, n$, and we use a transition probability notation. We denote states by the symbol x and successor states by the symbol y. The control/action is denoted by u, and is constrained to take values in a given finite constraint set $U(x)$, which may depend on the current state x. The use of a control u at state x specifies the transition probability $p_{xy}(u)$ to the next state y, at a cost $g(x, u, y)$.

A policy $\pi = \{\mu_0, \mu_1, \ldots\}$ is a sequence of functions from state to control that satisfies the control constraint, i.e., $\mu_k(x) \in U(x)$ for all x and k. Given a policy π and an initial state x_0, the system becomes a Markov chain whose generated trajectory under π, denoted $\{x_0, x_1, \ldots\}$, has a well-defined probability distribution. The corresponding total expected cost is

$$J_\pi(x_0) = \lim_{N\to\infty} E\left\{ \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k), x_{k+1}\big) \;\Big|\; x_0,\, \pi \right\}, \qquad x_0 = 1, \ldots, n,$$

where $\alpha < 1$ is the discount factor. The above expected value is taken with respect to the joint distribution of the states $x_1, x_2, \ldots$, conditioned on the initial state being x_0 and the use of π. As earlier, the optimal cost starting from a state x, i.e., the minimum of $J_\pi(x)$ over all policies π, is denoted by $J^*(x)$. We will view J^* as the vector of the n-dimensional space $\Re^n$ that has components $J^*(1), \ldots, J^*(n)$.†

The cost of a policy μ starting from state x is given by

$$J_\mu(x_0) = \lim_{N\to\infty} E\left\{ \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu(x_k), x_{k+1}\big) \;\Big|\; x_0,\, \mu \right\}, \qquad x_0 = 1, \ldots, n,$$

and J_μ is viewed as the vector in $\Re^n$ that has components $J_\mu(1), \ldots, J_\mu(n)$. In terms of our abstract notation, for each policy μ, the Bellman operator T_μ maps a vector $J \in \Re^n$ to the vector $T_\mu J \in \Re^n$ that has

† In our notation, $\Re^n$ is the n-dimensional Euclidean space and all vectors in $\Re^n$ are viewed as column vectors. Moreover, all vector inequalities $J \le J'$ are meant be componentwise, i.e., $J(x) \le J'(x)$ for all $x = 1, \ldots, n$.

components

$$(T_\mu J)(x) = \sum_{y=1}^{n} p_{xy}\big(\mu(x)\big)\big(g(x,\mu(x),y) + \alpha J(y)\big), \qquad x = 1,\ldots,n. \tag{5.1}$$

The Bellman operator $T : \Re^n \mapsto \Re^n$ is given by

$$(TJ)(x) = \min_{u\in U(x)} \sum_{y=1}^{n} p_{xy}(u)\big(g(x,u,y) + \alpha J(y)\big), \qquad x = 1,\ldots,n. \tag{5.2}$$

It is well-known that for discounted problems, the operators T_μ and T are sup-norm contractions, so J_μ is the unique solution of Bellman's equation $J = T_\mu J$, so that

$$J_\mu(x) = \sum_{y=1}^{n} p_{xy}\big(\mu(x)\big)\big(g(x,\mu(x),y) + \alpha J_\mu(y)\big), \qquad x = 1,\ldots,n. \tag{5.3}$$

Similarly, J^* is the unique solution of Bellman's equation $J = TJ$, so that

$$J^*(x) = \min_{u\in U(x)} \sum_{y=1}^{n} p_{xy}(u)\big(g(x,u,y) + \alpha J^*(y)\big), \qquad x = 1,\ldots,n. \tag{5.4}$$

A consequence of this is that the following optimality conditions hold

$$T_\mu J^* = TJ^* \qquad \text{if and only if} \qquad \mu \text{ is optimal,} \tag{5.5}$$

$$T_\mu J_\mu = TJ_\mu \qquad \text{if and only if} \qquad \mu \text{ is optimal.} \tag{5.6}$$

The contraction property also implies that the VI algorithms

$$J^{k+1} = T_\mu J^k, \qquad J^{k+1} = TJ^k$$

generate sequences $\{J^k\}$ that converge to J_μ and J^*, respectively, from any starting vector $J^0 \in \Re^n$.

As discussed in Section 3, in the PI algorithm, the current policy μ is improved by finding $\tilde\mu$ that satisfies

$$T_{\tilde\mu} J_\mu = TJ_\mu$$

[i.e., by minimizing for all x in the right-hand side of Eq. (5.2) with J_μ in place of J]. The improved policy $\tilde\mu$ is evaluated by solving the linear $n \times n$ system of equations $J_{\tilde\mu} = T_{\tilde\mu} J_{\tilde\mu}$, and then $(J_{\tilde\mu}, \tilde\mu)$ becomes the new cost vector-policy pair, which is used to start a new iteration. Thus the PI algorithm starts with a policy μ^0 and generates a sequence $\{\mu^k\}$ according to

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k}, \qquad T_{\mu^{k+1}} J_{\mu^k} = TJ_{\mu^k}. \tag{5.7}$$

We now introduce an on-line variant of PI, which starts at time 0 with a state-policy pair (x_0, μ^0) and generates on-line a sequence of state-policy pairs (x_k, μ^k). We view x_k as the current state of a system

operating on line under the influence of the policies $\mu^1, \mu^2, \ldots$. In our algorithm, μ^{k+1} *may differ from* μ^k *only at state* x_k. The control $\mu^{k+1}(x_k)$ and the state x_{k+1} are generated as follows:

We consider the Q-factors

$$Q_{\mu^k}(x_k, u) = \sum_{y=1}^{n} p_{x_k y}(u)\big(g(x_k, u, y) + \alpha J_{\mu^k}(y)\big), \tag{5.8}$$

and we select the control $\mu^{k+1}(x_k)$ from within the constraint set $U(x_k)$ with sufficient accuracy to satisfy the following sequential improvement condition

$$Q_{\mu^k}\big(x_k, \mu^{k+1}(x_k)\big) \leq J_{\mu^k}(x_k), \tag{5.9}$$

with strict inequality whenever this is possible.† For $x \neq x_k$ the policy controls are not changed:

$$\mu^{k+1}(x) = \mu^k(x) \quad \text{for all } x \neq x_k.$$

The next state x_{k+1} is generated randomly according to the transition probabilities $p_{x_k x_{k+1}}\big(\mu^{k+1}(x_k)\big)$.

We first show that the current policy is monotonically improved, i.e., that

$$J_{\mu^{k+1}}(x) \leq J_{\mu^k}(x), \qquad \text{for all } x \text{ and } k,$$

with strict inequality for $x = x_k$ (and possibly other values of x) if $\min_{u \in U(x_k)} Q_{\mu^k}(x_k, u) < J_{\mu^k}(x_k)$.

To prove this, we note that the policy update is done under the condition (5.9). By using the monotonicity of $T_{\mu^{k+1}}$, we have for all $\ell \geq 1$,

$$T^{\ell+1}_{\mu^{k+1}} J_{\mu^k} \leq T^{\ell}_{\mu^{k+1}} J_{\mu^k} \leq J_{\mu^k}, \tag{5.11}$$

so by taking the limit as $\ell \to \infty$ and by using the convergence property of VI ($T^{\ell}_{\mu^{k+1}} J \to J_{\mu^{k+1}}$ for any J), we obtain $J_{\mu^{k+1}} \leq J_{\mu^k}$. Moreover, the algorithm selects $\mu^{k+1}(x_k)$ so that

$$(T_{\mu^{k+1}} J_{\mu^k})(x_k) = Q_{\mu^k}(x_k, u_k) < J_{\mu^k}(x_k) \qquad \text{if } \min_{u \in U(x_k)} Q_{\mu^k}(x_k, u) < J_{\mu^k}(x_k),$$

[cf. Eq. (5.10)], so that by using Eq. (5.11), we have $J_{\mu^{k+1}}(x_k) < J_{\mu^k}(x_k)$.

† By this we mean that if $\min_{u \in U(x_k)} Q_{\mu^k}(x_k, u) < J_{\mu^k}(x_k)$ we select a control u_k that satisfies

$$Q_{\mu^k}(x_k, u_k) < J_{\mu^k}(x_k), \tag{5.10}$$

and set $\mu^{k+1}(x_k) = u_k$, and otherwise we set $\mu^{k+1}(x_k) = \mu^k(x_k)$ [so Eq. (5.9) is satisfied]. Such a control selection may be obtained by a number of schemes, including brute force calculation and random search based on Bayesian optimization. The needed values of the Q-factor Q_{μ^k} and cost J_{μ^k} may be obtained in several ways, depending on the problem at hand, including by on-line simulation.

Local Optimality

We next discuss the convergence and optimality properties of the algorithm. We introduce a definition of local optimality of a policy, whereby the policy selects controls optimally only within a subset of states.

Given a subset of states X and a policy μ, we say that μ *is locally optimal over* X if μ is optimal for the problem where the control is restricted to take the value $\mu(x)$ at the states $x \notin X$, and is allowed to take any value $u \in U(x)$ at the states $x \in X$.

Roughly speaking, μ is locally optimal over X, if μ is acting optimally within X, but under the (incorrect) assumption that once the state of the system gets to a state x outside X, there will be no option to select control other than $\mu(x)$. Thus if the choices of μ outside of X are poor, its choices within X may also be poor.

Mathematically, μ is locally optimal over X if

$$J_\mu(x) = \min_{u \in U(x)} \sum_{y=1}^{n} p_{xy}(u)\big(g(x,u,y) + \alpha J_\mu(y)\big), \qquad \text{for all } x \in X,$$

$$J_\mu(x) = \sum_{y=1}^{n} p_{xy}\big(\mu(x)\big)\Big(g\big(x,\mu(x),y\big) + \alpha J_\mu(y)\Big), \qquad \text{for all } x \notin X,$$

which can be written compactly as

$$(T_\mu J_\mu)(x) = (TJ_\mu)(x), \qquad \text{for all } x \in X. \tag{5.12}$$

Note that this is different than (global) optimality of μ, which holds if and only if the above condition holds for all $x = 1, \ldots, n$, rather than just for $x \in X$ [cf. Eq. (5.6)]. However, it can be seen that a (globally) optimal policy is also locally optimal within any subset of states.

Our main convergence result is the following.

Proposition 5.1: Let $\overline{X}$ be the subset of states that are repeated infinitely often within the sequence $\{x_k\}$. Then the corresponding sequence $\{\mu^k\}$ converges finitely to some policy $\overline{\mu}$ in the sense that $\mu^k = \overline{\mu}$ for all k after some index $\overline{k}$. Moreover $\overline{\mu}$ is locally optimal within $\overline{X}$, while $\overline{X}$ is invariant under $\overline{\mu}$, in the sense that

$$p_{xy}\big(\overline{\mu}(x)\big) = 0 \qquad \text{for all } x \in \overline{X} \text{ and } y \notin \overline{X}.$$

Proof: The cost function sequence $\{J_{\mu^k}\}$ is monotonically nondecreasing, as shown earlier. Moreover, the number of policies μ is finite in view of the finiteness of the state and control spaces. Therefore, the number

of corresponding functions J_μ is also finite, so J_{μ^k} converges in a finite number of steps to some $\bar{J}$, which in view of the algorithm's construction [selecting $u_k = \mu^k(x_k)$ if $\min_{u \in U(x_k)} Q_{\mu^k}(x_k, u) = J_{\mu^k}(x_k)$; cf. Eq. (5.10)], implies that μ^k will remain unchanged at some $\overline{\mu}$ with $J_{\overline{\mu}} = \bar{J}$ after some sufficiently large k.

We will show that the local optimality condition (5.12) holds for $X = \overline{X}$ and $\mu = \overline{\mu}$. In particular, we have $x_k \in \overline{X}$ and $\mu^k = \overline{\mu}$ for all k greater than some index, while for every $x \in \overline{X}$, we have $x_k = x$ for infinitely many k. It follows that for all $x \in \overline{X}$,

$$Q_{\overline{\mu}}\big(x, \overline{\mu}(x)\big) = J_{\overline{\mu}}(x), \tag{5.13}$$

while by the construction of the algorithm,

$$Q_{\overline{\mu}}\big(x, u\big) \geq J_{\overline{\mu}}(x), \qquad \text{for all } u \in U(x), \tag{5.14}$$

since the reverse would imply that $\mu^{k+1}(x) \neq \mu^k(x)$ for infinitely many k [cf. Eq. (5.10)]. Condition (5.13) can be written as $J_{\overline{\mu}}(x) = (T_{\overline{\mu}} J_{\overline{\mu}})(x)$ for all $x \in \overline{X}$, and combined with Eq. (5.14), implies that $(T_{\overline{\mu}} J_{\overline{\mu}})(x) = (T J_{\overline{\mu}})(x)$ for all $x \in \overline{X}$. This is the local optimality condition (5.12) with $X = \overline{X}$ and $\mu = \overline{\mu}$.

To show that $\overline{X}$ is invariant under $\overline{\mu}$, we argue by contradiction: if this were not so, there would exist a state $x \in \overline{X}$ and a state $y \notin \overline{X}$ such that

$$p_{xy}\big(\overline{\mu}(x)\big) > 0,$$

implying that y would be generated following the occurrence of x infinitely often within the sequence $\{x_k\}$, and hence would have to belong to $\overline{X}$ (by the definition of $\overline{X}$). **Q.E.D.**

Note an implication of the invariance property of the set $\overline{X}$ shown in the preceding proposition. We have that $\overline{\mu}$ is (globally) optimal under the assumption that for every policy there does not exist any strict subset of states that is invariant.

A Counterexample to Global Optimality

The following deterministic example shows that the policy $\overline{\mu}$ obtained by the algorithm need not be (globally) optimal. † Here there are three states 1, 2, and 3. From state 1 we can go to state 2 at cost 1, and to state 3 at cost 0, from state 2 we can go to states 1 and 3 at cost 0, and from state 3 we can go to state 2 at cost 0 or stay in 3 at a high cost (say 10). The discount factor is $\alpha = 0.9$. Then it can be verified that the optimal policy is

$$\mu^*(1) : \text{Go to 3}, \qquad \mu^*(2) : \text{Go to 3}, \qquad \mu^*(3) : \text{Go to 2},$$

† We thank Yuchao Li for constructing this example and providing computational support for other aspects of the paper.

with optimal costs

$$J^*(1) = J^*(2) = J^*(3) = 0,$$

while the policy

$$\overline{\mu}(1) : \text{Go to } 2, \qquad \overline{\mu}(2) : \text{Go to } 1, \qquad \overline{\mu}(3) : \text{Stay at } 3,$$

is strictly suboptimal, but is locally optimal over the set of states $\overline{X} = \{1, 2\}$. Moreover our on-line PI algorithm, starting from state 1 and the policy $\mu^0 = \overline{\mu}$, oscillates between the states 1 and 2, and leaves the policy μ^0 unchanged. Note also that $\overline{X}$ is invariant under $\overline{\mu}$, consistently with Prop. 5.1.

On-Line Variants of Policy Iteration with Global Optimality Properties

To address the local versus global convergence issue illustrated by the preceding example, we consider an alternative scheme, whereby in addition to u_k, we generate an additional control at a randomly chosen state $\overline{x}_k \neq x_k$.† In particular, assume that at each time k, in addition to u_k and x_{k+1} that are generated according to Eq. (5.10), the algorithm generates randomly another state $\overline{x}_k$ (all states are selected with positive probability), performs a policy improvement operation at that state as well, and modifies accordingly $\mu^{k+1}(\overline{x}_k)$. Thus, in addition to a policy improvement operation at each state within the generated sequence $\{x_k\}$, there is an additional policy improvement operation at each state within the randomly generated sequence $\{\overline{x}_k\}$.

Because of the random mechanism of selecting $\overline{x}_k$, it follows that at every state there will be a policy improvement operation infinitely often, which implies that the policy $\overline{\mu}$ ultimately obtained is (globally) optimal. Note also that *we may view the random generation of the sequence* $\{\overline{x}_k\}$ *as a form of exploration*. The probabilistic mechanism for generating the random sequence $\{\overline{x}_k\}$ may be guided by some heuristic reasoning, which aims to explore states with high cost improvement potential.

6. CONCLUDING REMARKS

While the ideas of approximation in value space, rollout, and PI have a long history, their significance has been highlighted by the success of AlphaZero and the earlier, but just as impressive, TD-Gammon program. Both programs were trained off-line extensively using sophisticated approximate PI algorithms and neural networks. Yet the players obtained off-line were greatly improved by on-line play, as we have discussed.

We have argued that this performance enhancement by on-line play defines a new paradigm for decision and control, which is couched on the AlphaZero/TD-Gammon design principles: on-line decision making,

† It is also possible to choose multiple additional states at time k for a policy improvement operation, and this is well-suited for the use of parallel computation.

using multistep optimization, approximation in value space, and rollout. There is an additional benefit of policy improvement by approximation in value space, not observed in the context of games (which have stable rules and environment). It is well-suited for on-line replanning and changing problem parameters, as in the context of indirect adaptive control, and also MPC, which in fact embodies several of the AlphaZero/TD-Gammon design ideas.

In this paper, we have aimed to provide the mathematical framework, analysis, and insights (often based on visualization), which facilitate the use of on-line decision making on top of off-line training. In particular, through a unified abstract DP analysis, we have shown that the principal ideas of approximation in value space and rollout apply very broadly to deterministic and stochastic optimal control problems, involving both discrete and continuous search spaces. These ideas can be effectively integrated with adaptive control, MPC, and other important methodologies such as decentralized and multiagent control, discrete and Bayesian optimization, neural network-based value and policy approximations, and heuristic algorithms for discrete optimization, as we have discussed in greater detail in the books [Ber19a] and [Ber20a]. We have also emphasized that while the ideas of on-line play and off-line training are implicit in several decision and control contexts, and particularly MPC, much remains to be gained by a more systematic view of the dichotomy between off-line training and on-line play, and by its incorporation into control system design contexts.

A key idea of this paper is the interpretation of approximation in value space with one-step lookahead as a step of Newton's method. This idea has been known for a long time within the more restrictive contexts of policy iteration and rollout. The extensions of this idea, including multistep lookahead and truncated rollout, which are provided in this paper, aim to promote the view that Newton's method and other classical algorithms, such as Newton-SOR, are central conceptual elements of the RL methodology.

A major supplementary idea of this paper is our interpretation of off-line training of policies and cost approximations as means for enhancement of the initial condition of the Newton step. Among others, this interpretation supports the idea that the Newton step/on-line player is the key determinant of the overall scheme's performance, and that the initial condition adjustment/off-line training plays a subsidiary role.

Finally, let us note that while our focus in this paper has been on infinite horizon problems, approximation in value space and rollout can be applied to finite horizon problems as well, and can be similarly interpreted in terms of Newton's method. One way to approach finite horizon problems analytically is to convert them to infinite horizon stochastic shortest path problems with a termination state that corresponds to the end of the horizon. Once this is done, the ideas of the present paper can be applied to provide insight on the connections between approximation in value space, rollout, and Newton's method. In particular, the ideas of this paper find application beyond the infinite horizon DP context, and apply to the solution of classical discrete and combinatorial optimization problems, which can be transformed to sequential finite

horizon optimization problems (see [BTW97]). Rollout has been used with great success for such problems in the past.

In particular, the book [Ber20a] contains many examples of application of rollout to discrete optimization and provides references to many works spanning the period from the late 90s to the present. These works discuss variants and problem-specific adaptations of rollout algorithms for a broad variety of practical problems, and consistently report favorable computational experience. The size of the cost improvement over the base policy is often impressive, evidently owing to the fast convergence rate of Newton's method that underlies rollout. Moreover these works illustrate some of the other important advantages of rollout: reliability, simplicity, suitability for on-line replanning, and the ability to interface with other RL techniques, such as neural network training, which can be used to provide suitable base policies and/or approximations to their cost functions.

7. REFERENCES

[AsH95] Aström, K. J., and Hagglund, T., 1995. PID Controllers: Theory, Design, and Tuning, Instrument Society of America, Research Triangle Park, NC.

[AsH06] Aström, K. J., and Hagglund, T., 2006. Advanced PID Control, Instrument Society of America, Research Triangle Park, N. C.

[AsM10] Aström, K. J., and Murray, R. M., 2010. Feedback Systems, Princeton Univ. Press.

[AsW08] Aström, K. J., and Wittenmark, B., 2008. Adaptive Control, Dover Books; also Prentice-Hall, Englewood Cliffs, N. J, 1994.

[BBM17] Borrelli, F., Bemporad, A., and Morari, M., 2017. Predictive Control for Linear and Hybrid Systems, Cambridge Univ. Press, Cambridge, UK.

[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.

[BMZ09] Bokanowski, O., Maroso, S., and Zidani, H., 2009. "Some Convergence Results for Howard's Algorithm," SIAM J. on Numerical Analysis, Vol. 47, pp. 3001-3026.

[BSW99] Beard, R. W., Saridis, G. N., and Wen, J. T., 1998. "Approximate Solutions to the Time-Invariant Hamilton-Jacobi-Bellman Equation," J. of Optimization Theory and Applications, Vol. 96, pp. 589-626.

[BTW97] Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C., 1997. "Rollout Algorithms for Combinatorial Optimization," Heuristics, Vol. 3, pp. 245-262.

[BeR71] Bertsekas, D. P., and Rhodes, I. B., 1971. "On the Minimax Reachability of Target Sets and Target Tubes," Automatica, Vol. 7, pp. 233-247.

[BeS78] Bertsekas, D. P., and Shreve, S. E., 1978. Stochastic Optimal Control: The Discrete Time Case, Academic Press, N. Y.; republished by Athena Scientific, Belmont, MA, 1996 (can be downloaded from the author's website).

[BeT89] Bertsekas, D. P., and Tsitsiklis, J. N., 1989. Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Engl. Cliffs, N. J. (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 Research, Vol. 16, pp. 580-595.

[BeT96] Bertsekas, D. P., and Tsitsiklis, J. N., 1996. Neuro-Dynamic Programming, Athena Scientific, Belmont, MA.

[BeY10] Bertsekas, D. P., and Yu, H., 2010. "Asynchronous Distributed Policy Iteration in Dynamic Pro-

gramming," 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.

[Bea95] Beard, R. W., 1995. Improving the Closed-Loop Performance of Nonlinear Systems, Ph.D. Thesis, Rensselaer Polytechnic Institute.

[Ber71] Bertsekas, D. P., 1971. "Control of Uncertain Systems With a Set-Membership Description of the Uncertainty," Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA (can be downloaded from the author's website).

[Ber72] Bertsekas, D. P., 1972. "Infinite Time Reachability of State Space Regions by Using Feedback Control," IEEE Trans. Automatic Control, Vol. AC-17, pp. 604-613.

[Ber82] Bertsekas, D. P., 1982. "Distributed Dynamic Programming," IEEE Trans. Aut. 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.

[Ber05] Bertsekas, D. P., 2005. "Dynamic Programming and Suboptimal Control: A Survey from ADP to MPC," European J. of Control, Vol. 11, pp. 310-334.

[Ber12] Bertsekas, D. P., 2012. Dynamic Programming and Optimal Control, Vol. II, 4th Ed., Athena Scientific, Belmont, MA.

[Ber16] Bertsekas, D. P., 2016. Nonlinear Programming, Athena Scientific, Belmont, MA.

[Ber17a] Bertsekas, D. P., 2017. Dynamic Programming and Optimal Control, Vol. I, 4th Ed., Athena Scientific, Belmont, MA.

[Ber17b] Bertsekas, D. P., 2017. "Value and Policy Iteration in Deterministic Optimal Control and Adaptive Dynamic Programming," IEEE Trans. on Neural Networks and Learning Systems, Vol. 28, pp. 500-509.

[Ber18a] Bertsekas, D. P., 2018. Abstract Dynamic Programming, 2nd Ed., Athena Scientific, Belmont, MA (can be downloaded from the author's website).

[Ber18b] Bertsekas, D. P., 2018. "Feature-Based Aggregation and Deep Reinforcement Learning: A Survey and Some New Implementations," Lab. for Information and Decision Systems Report, 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.

[Ber19a] Bertsekas, D. P., 2019. Reinforcement Learning and Optimal Control, Athena Scientific, Belmont,

MA.

[Ber19b] Bertsekas, D. P., 2019. "Multiagent Rollout Algorithms and Reinforcement Learning," arXiv preprint arXiv:1910.00120.

[Ber19c] 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 Value Iteration Algorithms in Dynamic Programming and Reinforcement Learning," arXiv preprint, arxiv.org/abs/2005.01627; Results in Control and Optimization J., Vol. 1, 2020.

[Ber21a] Bertsekas, D. P., 2021. "On-Line Policy Iteration for Infinite Horizon Dynamic Programming," arXiv preprint arXiv:2106.00746.

[Ber21b] Bertsekas, D. P., 2021. "Multiagent Reinforcement Learning: Rollout and Policy Iteration," IEEE/CAA J. of Automatica Sinica, Vol. 8, pp. 249-271.

[Ber21c] Bertsekas, D. P., 2021. "Distributed Asynchronous Policy Iteration for Sequential Zero-Sum Games and Minimax Control," arXiv preprint arXiv:2107.10406, July 2021.

[Bod20] Bodson, M., 2020. Adaptive Estimation and Control, Independently Published.

[CaB07] Camacho, E. F., and Bordons, C., 2007. Model Predictive Control, 2nd Ed., Springer, New York, N. Y.

[KoC16] Kouvaritakis, B., and Cannon, M., 2016. Model Predictive Control: Classical, Robust and Stochastic, Springer, N. Y.

[GSD06] Goodwin, G., Seron, M. M., and De Dona, J. A., 2006. Constrained Control and Estimation: An Optimisation Approach, Springer, N. Y.

[GoS84] Goodwin, G. C., and Sin, K. S. S., 1984. Adaptive Filtering, Prediction, and Control, Prentice-Hall, Englewood Cliffs, N. J.

[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.

[Hyl11] Hylla, T., 2011. Extension of Inexact Kleinman-Newton Methods to a General Monotonicity Preserving Convergence Theory, Ph.D. Thesis, Univ. of Trier.

[IoS96] Ioannou, P. A., and Sun, J., 1996. Robust Adaptive Control, Prentice-Hall, Englewood Cliffs, N. J.

[JiJ17] Jiang, Y., and Jiang, Z. P., 2017. Robust Adaptive Dynamic Programming, J. Wiley, N. Y.

[KKK95] Krstic, M., Kanellakopoulos, I., Kokotovic, P., 1995. Nonlinear and Adaptive Control Design, J. Wiley, N. Y.

[KKK20] Kalise, D., Kundu, S., and Kunisch, K., 2020. "Robust Feedback Control of Nonlinear PDEs by Numerical Approximation of High-Dimensional Hamilton-Jacobi-Isaacs Equations." SIAM J. on Applied Dynamical Systems, Vol. 19, pp. 1496-1524.

[Kle68] Kleinman, D. L., 1968. "On an Iterative Technique for Riccati Equation Computations," IEEE Trans. Automatic Control, Vol. AC-13, pp. 114-115.

[Kok91] Kokotovic, P. V., ed., 1991. Foundations of Adaptive Control, Springer.

[KuK21] Kundu, S., and Kunisch, K., 2021. "Policy Iteration for Hamilton-Jacobi-Bellman Equations with Control Constraints," Computational Optimization and Applications, pp. 1-25.

[KuV86] Kumar, P. R., and Varaiya, P. P., 1986. Stochastic Systems: Estimation, Identification, and Adaptive Control, Prentice-Hall, Englewood Cliffs, N. J.

[LJM21] Li, Y., Johansson, K. H., Martensson, J., and Bertsekas, D. P., 2021. "Data-Driven Rollout for Deterministic Optimal Control," arXiv preprint arXiv:2105.03116.

[LWW17] Liu, D., Wei, Q., Wang, D., Yang, X., and Li, H., 2017. Adaptive Dynamic Programming with Applications in Optimal Control, Springer, Berlin.

[LaW13] Lavretsky, E., and Wise, K., 2013. Robust and Adaptive Control with Aerospace Applications, Springer.

[MRR00] Mayne, D., Rawlings, J. B., Rao, C. V., and Scokaert, P. O. M., 2000. "Constrained Model Predictive Control: Stability and Optimality," Automatica, Vol. 36, pp. 789-814.

[MVB20] Magirou, E. F., Vassalos, P., and Barakitis, N., 2020. "A Policy Iteration Algorithm for the American Put Option and Free Boundary Control Problems," J. of Computational and Applied Mathematics, vol. 373, p. 112544.

[Mac02] Maciejowski, J. M., 2002. Predictive Control with Constraints, Addison-Wesley, Reading, MA.

[May14] Mayne, D. Q., 2014. "Model Predictive Control: Recent Developments and Future Promise," Automatica, Vol. 50, pp. 2967-2986.

[NaA12] Narendra, K. S., and Annaswamy, A. M., 2012. Stable Adaptive Systems, Courier Corporation.

[OrR70] Ortega, J. M., and Rheinboldt, W. C., 2000. Iterative Solution of Nonlinear Equations in Several Variables, Society for Industrial and Applied Mathematics.

[PoA69] Pollatschek, M., and Avi-Itzhak, B., 1969. "Algorithms for Stochastic Games with Geometrical Interpretation," Management Science, Vol. 15, pp. 399-413.

[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," Math. of Operations Research, Vol. 4, pp. 60-69.

[RMD17] Rawlings, J. B., Mayne, D. Q., and Diehl, M. M., 2017. Model Predictive Control: Theory, Computation, and Design, 2nd Ed., Nob Hill Publishing (updated in 2019 and 2020).

[Rek64] Rekasius, Z. V., 1964. "Suboptimal Design of Intentionally Nonlinear Controllers," IEEE Trans. on Automatic Control, Vol. 9, pp. 380-386.

[RoB17] Rosolia, U., and Borrelli, F., 2017. "Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework," IEEE Trans. on Automatic Control, Vol. 63, pp. 1883-1896.

[RoB19] Rosolia, U., and Borrelli, F., 2019. "Sample-Based Learning Model Predictive Control for Linear Uncertain Systems," 58th Conference on Decision and Control (CDC), pp. 2702-2707.

[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.

[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.

[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.

[SaB11] Sastry, S., and Bodson, M., 2011. Adaptive Control: Stability, Convergence and Robustness, Courier Corporation.

[SaL79] Saridis, G. N., and Lee, C.-S. G., 1979. "An Approximation Theory of Optimal Control for Trainable Manipulators," IEEE Trans. Syst., Man, Cybernetics, Vol. 9, pp. 152-159.

[SlL91] Slotine, J.-J. E., and Li, W., Applied Nonlinear Control, Prentice-Hall, Englewood Cliffs, N. J.

[SaR04] Santos, M. S., and Rust, J., 2004. "Convergence Properties of Policy Iteration," SIAM J. on Control and Optimization, Vol. 42, pp. 2094-2115.

[SuB18] Sutton, R., and Barto, A. G., 2018. Reinforcement Learning, 2nd Ed., MIT Press, Cambridge, MA.

[TeG96] Tesauro, G., and Galperin, G. R., 1996. "On-Line Policy Improvement Using Monte Carlo Search," NIPS, Denver, CO.

[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.

[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.

[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.